计算机基础教育丛书

COBOL 语 言（上册）

修订版

谭浩强 编著

清华大学出版社

目　录

第一章　COBOL 语言概述

§1.1　COBOL 语言的发展概况

COBOL 是 Common Business Oriented Language(通用商业语言)的缩写。实际上，COBOL 不仅是商业数据处理的理想语言，而且广泛应用于数据管理领域，例如财会工作、统计报表、计划编制、情报检索、人事管理等。因此 COBOL 语言也被称为"用于管理的语言"。

在计算机的应用领域中，数据处理(data processing)是应用最广泛的一个领域。数据处理的日益广泛应用要求人们设计出能满足实际数据处理工作中各种要求的一种计算机语言。COBOL 语言就是在这种形势下应运而生的。

1959 年 5 月，美国国防部召开了一个有政府机关、企业、计算机厂家代表参加的会议，各方面都认为有必要设计出一种数据处理专用的计算机语言。会上确定了常设机构，以研究这种语言。这个会议称为 CODASYL(Conference on Data Systems Languages)，意为数据系统语言会议。1959 年 12 月提出了世界上第一个 COBOL 语言文本，次年 4 月由美国政府印刷局正式发表，因此称 COBOL-60。后来进一步扩充和完善，出现了 COBOL-61，扩展 COBOL-61。它们为后来的版本提供了基础。

1965 年美国出现了更完善的版本，即 COBOL-65，但直到 1968 年 8 月才由美国国家标准化协会 ANSI(American National Standard Institute)通过批准了这个语言的标准版本，作为各厂家的依据。这就是 ANSI COBOL X3.23-1968。1972 年国际标准化组织 ISO (International Standard Organization)决定把它作为 ISO COBOL-72 国际标准 COBOL 文本，该文本已为美、英、法、日、苏等 21 个会员国承认。

1974 年，美国 ANSI 对 COBOL-68 作了修改扩充，发表了 ANSI COBOL X3.23-1974 文本。1978 年 ISO 宣布 ANSI COBOL X3.23-1974 作为国际标准文本，即 ISO COBOL-78。

标准版本的出现，为 COBOL 语言的推广应用创造了一个有利的条件。多年来各国计算机厂商都以 ISO COBOL 72 (即 ANSI COBOL 1968)或 ISO COBOL-78(即 ANSI COBOL 1974)作为设计软件的依据。几年前，美国又提出新的 COBOL 版本，即 COBOL 85。它在 ANSI COBOL 1974 的基础上，扩充了功能，适合于结构化程序设计。但目前国内大多数计算机系统还未配置 COBOL 85 编译系统，而仍普遍使用 ANSI COBOL 1974。因此本书的叙述仍然以 COBOL 1974 为基础，并在附录中给出 COBOL 85 的语法一览。掌握了 COBOL 1974 之后，再学习 COBOL 85 是不会有什么困难的。

应该说明，尽管 COBOL 标准化程度比较高；但各个计算机厂家在实现它时还是有一些差别的，在具体用计算机时应查阅该计算机系统的 COBOL 说明书。

§1.2 COBOL 语言的特点

COBOL 语言的主要特点有：

（一）最适于数据处理领域。所谓数据处理是指对大量数据的收集、统计、分类和加工。例如企业管理、库存管理、报表统计、帐目计算、信息情报检索等方面的应用都属于数据处理。

数据处理的特点是：算术计算量少而逻辑处理多；输入输出量大；数据间存在着一定的逻辑关系（数据项间有清晰的层次关系，例如职工工资中包括应发工资、扣除部分、实发工资几部分，应发工资又包括基本工资、附加工资等）；大量的分类排序（如按年龄大小排名单，按受教育程度分类……）；对打印报表要求较高、多样化等等。

在企业（如银行、商业、工厂）和其它部门（如领导机关、业务管理部门）的管理工作中，一般并无很复杂的计算公式，不要求太高深的数学基础，但是处理数据的量很大。

COBOL 正是针对数据处理要求而设计的。COBOL 所处理的问题具有数据繁多而运算简单的特点。COBOL 中也有加、减、乘、除、乘方等运算以及表达式的概念，但这些不是 COBOL 的重点。它的主要功能是描述数据结构和分析处理大批量的数据。

COBOL 对数据的处理过程，与人工处理的过程是相似的，即与人们的思维过程比较接近，因此，一般的管理人员是比较容易理解和掌握 COBOL 语言的。

（二）COBOL 比较接近于自然语言（指的是英语）。COBOL 程序看起来很像一篇用英语写的文章。例如，用 ADD A TO B 来表示 A+B⇒B（A 加 B，结果放在 B 中），用 MOVE C TO D 表示将变量 C 的值传送到变量 D 中。COBOL 大量采用普通英语词汇和句型，学过英语的人看 COBOL 程序感到通俗易懂。也就是说它的特点是：**成文自明**。看它的英文意思就可以大致懂得程序的含义。

（三）通用性强，由于 COBOL 语言的标准化程度较高。不同厂家生产的计算机系统所提供的 COBOL，是 COBOL 标准的全集或一个子集。一个计算机上的 COBOL 程序向另一计算机系统上移植，是比较容易实现的。

（四）COBOL 的结构严谨，层次分明。每个程序分四大部分（称为部，division），每个部下面又分为若干节（section），节下面又分为若干段（paragraph）。每一部分都有固定的程式。这个特点使初学者比较容易通过摹仿别人程序中的有关部分，从而较快地写出自己的程序。

（五）COBOL 的缺点是比较繁琐。如同中国古代的八股文一样，程序无论大小简繁，一律都要写齐四大部分，对每个部进行必要的定义和说明。因此源程序显得比较冗长。

据国外统计，在大、中型计算机系统上运行 COBOL 程序所占用的计算机时间为全部机时一半以上，超过了任何一种其它语言，是目前世界上使用得最多的一种计算机语言。

§1.3 最简单的 COBOL 程序介绍

为了使初学者从一开始就了解 COBOL 源程序的格式以及它的组成，建立起一个整体的概念，我们在这一节中先介绍两个最简单的 COBOL 源程序。

【例 1.1】

```
1     6   7   8          12
          IDEN  TIFICATION          DIVISION.(标识部)
          PROG  RAM-ID.        EXAM1.        (程序标识段)
          ENVI  RONMENT             DIVISION.(环境部)
          DATA                      DIVISION.(数据部)
          PROC  EDURE               DIVISION.(过程部)
          S.    DISPLAY       'THIS IS A COBOL PROGRAM.'
                STOP    RUN.
```

这个程序的目的是使计算机在指定的外部设备(终端显示器或打印机)上显示(或打印)出"THIS IS A COBOL PROGRAM。"这样一串字符,然后停止运行。这些操作是在"过程部"中指定的。程序倒数第二行的"S"是段名。在本例中过程部只包括一个段,即S段。在S段中有两个句子,每个句子以句点"."和空格结束。

【例1.2】 将A和B的值相加,其结果放在B中。

```
1     6   7   8          12
          IDEN  TIFICATION          DIVISION.(标识部)
          PROG  RAM ID.             EXAM2.   (程序标识段)
          ENVI  RONMENT             DIVISION.(环境部)
          DATA                      DIVISION.(数据部)
          WORK  ING STORAGE         SECTION.(工作单元节)
          77    PICTURE      IS     9(3).    (对A进行描述)
          77 ,  PICTURE      IS     9(3).    (对B进行描述)
          PROC  EDURE               DIVISION.(过程部)
          S.    ACCEPT       A                (输入A的值)
                ACCEPT       B                (输入B的值)
                ADD          A      TO  B     (A+B→B)
                DISPLAY      A,     B.        (显示A和B的值)
                STOP    RUN.                  (停止运行)
```

这个程序的名字叫"EXAM2"。与例1.1不同,在本例中数据部下面有一个WORKING-STORAGE SECTION(工作单元节,或称工作存储节),用它来描述程序中用到的中间工作单元。今有两个数据项A和B,用"PICTURE IS 9(3)"来说明(描述)A和B的类型是数值型的,"9"代表数值型,"(3)"代表数据长度为三位,即A和B的值是三位整数(有关数据描述将在以后详述,在此只要求大体知道它们的作用即可)。在过程部中,只有一个S段。在S段中有两个句子。每个句子以句点和空格为结束标志。第一个句子中包含四个语句,每个语句完成一个特定的操作。ACCEPT A和ACCEPT B是从指定的外部设备上先后接收两个数值给A和B(指定的外部设备可以是控制台或终端的键盘,也可以是读卡机或软磁

盘机，由具体的计算机系统规定）。例如，如果从键盘上打入“012”和“024”二个数值给计算机，则 A 的值为 12，B 的值为 24。第三个语句是加法语句，ADD A TO B 表示“将 A 的值加到 B 上面去”，即 A＋B⇒B，也就是 12＋24＝36，将 36 存放在 B 中，因此，B 的值为 36。第四个语句为显示语句，显示出 A 和 B 的值，在指定的外部设备上输出 A 和 B 的值（12 和 36）。至此，第一个句子完了。第二个句子是 STOP RUN，停止程序运行。

对这两个例子，可以暂时“不求甚解”，即只要求大体上知道它们的意思即可，有关各部分将在下面各章中详细说明。

§1.4 COBOL 程序的结构

1.4.1 部

从上面两个例子中可以看到，每一个程序都应包括以下四大部分，每个部（Division）有自己的部名，每一个部的作用如下：

IDENTIFICATION DIVISION	（标识部）主要用来指定源程序名字，也可以写入其它用作备忘的某些信息（如日期、作者等）。
ENVIRONMENT DIVISION	（环境部）指出程序中用到的数据文件名与计算机系统的设备的对应关系，即把某一文件名与一外部设备联系起来。此外还指定目标程序中使用的专门控制方法及程序所用内存区的大小等。
DATA DIVISION	（数据部）程序中所用到的全部数据（包括输入输出的数据和中间数据）都应在数据部中说明它们的类型和所占内存的情况。
PROCEDURE DIVISION	（过程部）用来给出程序要执行的指令，使计算机产生相应的操作。例如进行运算或其它处理。

在以上四个部分中，只有过程部是执行部分。计算机的任何一个操作都是由过程部中的指令给出的。因此，过程部是整个程序的核心部分，由它决定程序的每一步操作。也就是说，程序所预定的功能主要是靠这部分来完成的。前面三个部分是对过程部中用到的各文件、数据项和程序执行时的环境等作必要的描述和声明。例如，在过程部中指定要进行 A＋B 的操作，结果放在 B 中。则应在数据部中说明 A 和 B 是数值型的数据项，并说明它在计算机内存中的存储形式和所占内存的情况（例 1.2 中 A 和 B 在内存中各占三个字节。一个字节为八位（bit），这里‘位’指的是二进制位）。在执行到过程部中加法语句“ADD A TO B”时，就从数据部中所定义的数据项 A 和 B（它们代表内存中一定的存储单元）中把数值取出来进行相加，把结果再送回内存中名为 B 的存储单元中。

从这两个程序中可以看出：一个程序中四大部分缺一不可。即使有的部（例如例 1.1 中环境部）的下面并无具体内容，也要与上“部头”（或称“部首”），如 ENVIRONMENT DIVISION。

1.4.2 节和段

除标识部以外，在每一个部的“部头”的下面，可以有若干个节（SECTION），每一个节以“节头”作标识。每一节下面又可包括若干段（PARAGRAPH）。每一个段都有自己的名字（即段名）。如上节例 1.2 中，数据部下面有一个节，WORKING-STORAGE SECTION（工作单元节）。在标识部下面不设节，直接设段，如上节例中，PROGRAM-ID。就是“程序标识段”的段头。在它的后面写上由程序设计者确定的程序名（如 EXAM1 或 EXAM2），以便与其它程

序相区别。过程部下面可以设节，下面再设段，也可以直接设段。例 1.1 和例 1.2 中过程部下面都没有设节，它直接由一个段构成。段名是“S”。在一般简单的程序中，过程部内可以不设节，直接由段构成。只有复杂的程序才在过程部下设节，节下分段。环境部(ENVIRONMENT DIVISION)和数据部(DATA DIVISION)下面是设节的。

程序结构可以示意如下：

```
程序：
IDENTIFICATION DIVISION.
  段
  ⋮
  ⋮
ENVIRONMENT DIVISION.
  节
    段
    ⋮
  节
  ⋮
DATA DIVISION.
  节
    描述体
    ⋮
  ⋮
  ⋮
PROCEDURE DIVISION.
 (节)
  段
  ⋮
  ⋮
```

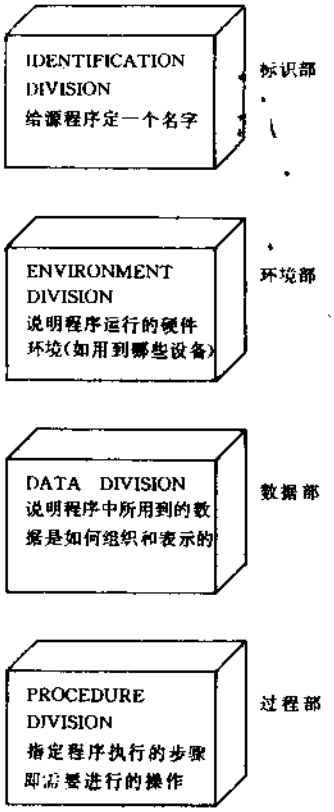


图 1.1

1.4.3 句子、语句和子句

在过程部中，每一段由若干个句子(Sentence)组成。一个句子是以句号加一个以上的空格来结束的。例如例 1.1 中 S 段由两个句子组成。第一个句子是：DISPLAY 'THIS IS A COBOL PROGRAM.', ",第二个句子是：“STOP RUN.”。注意每个句子最后都有一个句点和一个以上的空格。

句子又由语句(Statement)组成。例 1.2 中 S 段由两个句子组成。其中第一个句子包括四个语句(占四行)，在第四行的最后才有句点和空格(想要知道有几个句子只需数一下有几个后面跟空格的句点即可)。每一语句都是一条完整的指令，它们各自有一个相应的动词(Verb)，表示该语句指定计算机要进行的操作，如 ACCEPT(从指定的设备上接收某些值)，ADD(进行加法)，DISPLAY(在指定的外部设备上显示信息)。

一个句子可以只由一个语句组成，如例 1.2 的第二个句子 STOP RUN.，一个语句加一个句点后跟一个(多个)空格就成为一个句子。

在一个语句中又可以包含若干个子句(clause)，每一子句也有一个动词(但这个动词往往是可以省略的)，它指定某一方面特定的功能。

过程部中程序的结构如下：

部(Division)　　一部可包括若干节

节(Section)　　一节可包括若干段
段(Paragraph)　一段可包括若干句子
句子(Sentence)　一句子可包括若干语句
语句(Statement)指定计算机完成一定的操作
子句(Clause)指定完成某一方面的特定功能

除了过程部的语句可以包含子句外，在环境部和数据部中也可以出现子句，如 SELECT 子句，文件描述子句等。见 1.5.4 段。

1.4.4 描述体

在数据部中有若干节，每个节中有若干个描述体(Description entry，亦译作描述款目或描述款)，每个描述体又由若干个子句构成。

例如，例 1.2 中数据部(DATA DIVISION)中的工作单元节(WORKING-STORAGE SECTION)下面，有二行(以 77 打头的)，它们就是数据描述体，分别描述 A 和 B 的数据形式，说明 A 和 B 都是数值型的数据，而且是三位整数的数。以后我们还可以看到其它形式的描述体(如文件描述，记录描述等)。有关描述体的概念我们将在第四章中作详细介绍。

整个 COBOL 程序的结构可用图 1.2 表示：

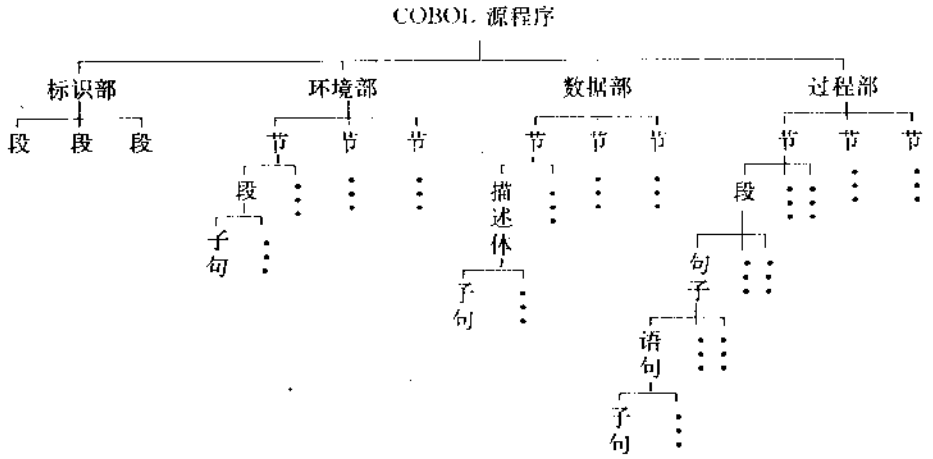


图 1.2

§1.5 COBOL 源程序的书写格式

COBOL 源程序必须严格地按规定的格式书写。其书写格式通常采用 ANSI 格式(American National Standard COBOL reference format)。有些计算机系统除了可以采用 ANSI 格式书写和输入源程序外，还可以采用终端格式(Terminal format)。如 VAX COBOL，在编译时使用不同命令以分别对这两种格式的源程序进行编译。

1.5.1 ANSI 书写格式

ANSI 源程序书写格式见图 1.3。

COBOL 程序纸每行有 80 列(每张程序纸包括多少行不作规定，一般为 20～25 行)。每

一行分为几个区：

标号区		续行区	A 区		B 区 (正文区)		注解	
1	6	7	8	11	12	72	73	80

图 1.3

1. 1~6 列，为"标号区"。可以填写 6 个数字。标号由程序编写者自定，一般用前 3 列表示页号，后 3 列表示在本页中的行号。如用 001015 表示程序的第一页第 15 行。标号应按由小到大的顺序，但不一定连续。标号区内可以写标号也可以不写标号(标号区空白)。标号对源程序的执行结果没有任何影响。标号只是为程序人员查阅程序时方便而设的。例如在一个长的程序中要找某一行，显然按标号查找是比较方便的。在程序编译时是按程序书写的顺序进行的，而不是按标号大小顺序进行的。

2. 第 7 列，是"续行标志区"。如果在第 7 列上写上连接符"－"，则表示本行是紧接在上一行的后面的。如上一行中有一个字 MOVE 未写完，只写了 MO 二个字符，在下一行的第 7 列上写"－"，然后在 12 列开始写 VE 二个字符，则表示 VE 是紧连在上一行 MO 之后的，等效于在上一行中写 MOVE，图 1.4 中①和②两种写法是等效的。但如一个字已写完，本应空一格再写下一个字，因上一行写不下而要求在下一行续写，此时不必在第七列上写"－"，而是使第七列空白，则下一行 12 列后的内容自动连在上一行之后，中间插入一个空格。见图 1.4③。

以上①②③④四种写法是等效的。第①种是将 VE 紧接在 MO 之后，二行连起来，组成

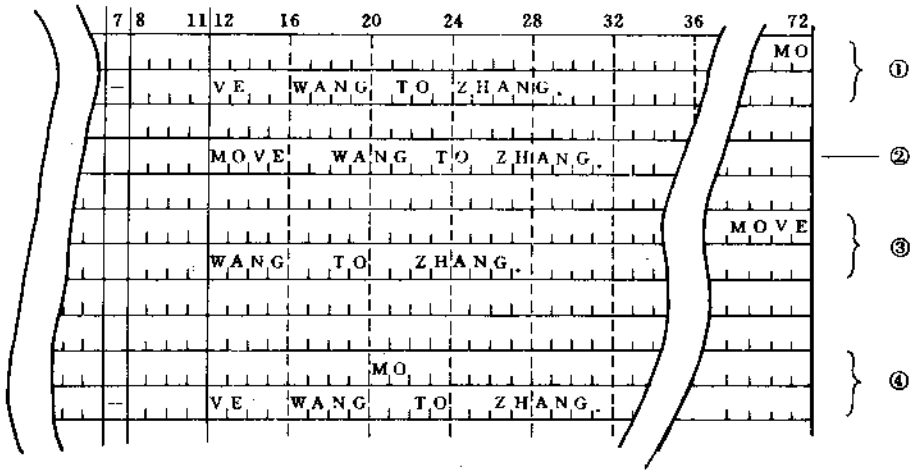


图 1.4

MOVE WANG TO ZHANG。第③种是把 WANG 接在 MOVE 之后，二者间自动插入空格，MOVE 和 WANG 分别是两个字。第①种写法中由于在输入"MO"后就按回车键结束了本

行，而在继续行的第七列中有“－”，因此，将 MO 与 VE 紧连，成 MOVE。因此，只有在上下二行“紧连”时才在下一行的第七列上加“－”。否则不要加‘－’。但应尽可能避免将一个字拆成二行，以免产生错误。

如果有一个用引号括起来的字符串（如例 1.1 中 DISPLAY 语句中的 'THIS IS A COBOL PROGRAM.'），上一行没写完，需下一行继续时，应在续行的第 7 列上写上“－”，然后在 12 列后写继续的部分，并在其第一个字符前加一个引号。如图 1.5 中的①和②是等效的。

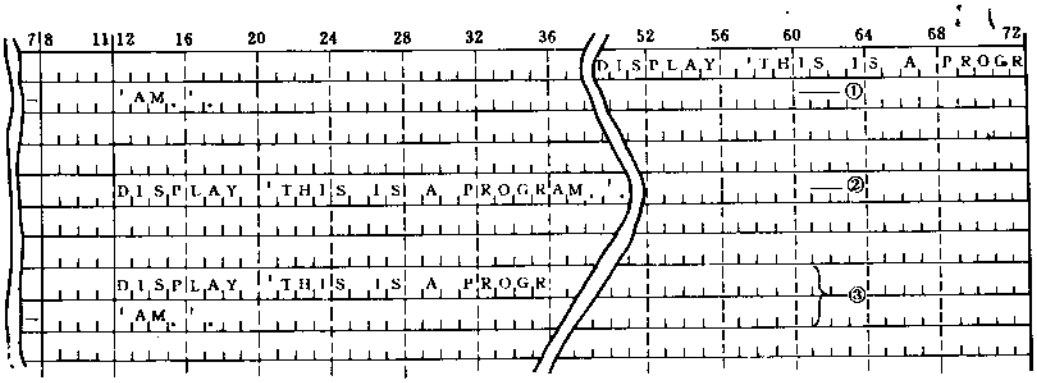


图 1.5

注意如果在上一行的最后一个字符后面有一个或多个空格，如图 1.5 的③中所示，在“PROGR”后面有多个空格，这个字符串中 PROGR 和 AM 之间就插入了许多个空格。因此③和①或②是不等效的。这是在使用字符串时所应注意的。

如果在第 7 列中不写“－”而写“＊”，则表示此行是注解行，即此行可由程序员任意写上自己所需的内容，以对程序（或程序的一部分）作说明。它对程序的执行不起任何影响，仅起一个注释作用（只给程序员自己作备忘之用，或使别人能看懂此程序而作必要的说明）。计算机在编译时不理会此行。只是在打印源程序清单时把此行原样印出。

3. 第 8～11 列，称为“A 区”，第 8 列称“A 区边界”。COBOL 规定，程序中有些内容，例如部头，节头，段头，层号 01，层号 77 以及文件描述符 FD 等应从 A 区开始书写（有关这些内容将在后面叙述）。从 A 区开始写，并不一定要从第 8 列开始，也可以从第 9 列，或第 10 列、第 11 列开始写。

4. 12～72 列，称“B 区”。写程序中的正文部分，例如过程部中的句子只能从 B 区开始写，而不能写到 A 区去。哪些内容应从 B 区开始，以后会陆续介绍的。

5. 73～80 列，为“注释区”。程序员如想对源程序的某些行作些简单说明，可写在这 8 列中，例如编号，或注明其作用。计算机在编译源程序时不理会这 8 列。它对程序的执行不起作用。因此写源程序时注意不应超过第 72 列，超过的部分在编译时将被舍弃。

注意，以上对 80 列的划分区，只是对源程序来说的，如果程序纸上写的不是源程序，而是输入的数据，则不受以上分区的限制（输入数据时可以从第 1 列用到第 80 列）。这一点有的初学者常易搞混。我们只是在这里预先指出这一点。关于输入数据的方法将在以后介绍。

图 1.6 表示一个 COBOL 源程序的书写格式的例子。

关于这个程序将在第四章中解释。

GOBOL 程序名称　　设计人　,共　　页,第　　页,　日期

```
IDENTIFICATION DIVISION.                                    (标识部)
PROGRAM-ID. EXAM.
ENVIRONMENT DIVISION.                                       (环境部)
INPUT-OUTPUT SECTION.                                       (输入-输出节)
FILE-CONTROL.                                               (文件控制段)
    SELECT ABC ASSIGN TO FILE1.                             (输入文件)
    SELECT XYZ ASSIGN TO FILE2.                             (输出文件)
DATA DIVISION.                                              (数据部)
FILE SECTION.                                               (文件节)
FD ABC LABEL RECORD IS STANDARD.                            (文件描述体)
01 A PIC X(80).
FD XYZ LABEL RECORD IS STANDARD.                            (文件描述体)
01 B.
   02 FILLER PIC X.
   02 C PIC X(136).
PROCEDURE DIVISION.                                         (过程部)
K.  OPEN INPUT ABC OUTPUT XYZ.                              (K段)
T.  READ ABC AT END GO TO J.                                (T段)
    MOVE A TO C.
    WRITE B AFTER ADVANCING 2 LINES.
    GO TO T.
J.  DISPLAY 'THE CARDS ARE FINISHED'.                       (J段)
    CLOSE ABC XYZ.
    STOP RUN.
```

图 1.6

在写程序时，应注意以下几点：

(1) 每个字符占程序纸中的一个格。

(2) 较早的COBOL版本规定所有字母都应大写，但现在使用的COBOL编译系统允许使用大写或小写字母，二者等价，但用引号括起来的字符串中的大写和小写是有区别的，二者不等价。

(3) 相邻的两个字(如COBOL的保留字或用户自己定义的名字)之间必须留一个以上的空格。

(4) 运算符(如加、减、乘、除、乘方)和等号左右两边必须各留一个空格。在过程部中左括号的左侧和右括号的右侧要留一空格，而内侧不必留空格。如：

```
A + (B + C) / D.
```

(5) 逗号、句号、分号的左边不能留空格，而右边应有空格。

(6)一个空格和多个空格作用相同，如：

```
MOVE A TO B
和 MOVE A    TO    B
和 MOVE A
  TO    B 等价。
```

但被引号括起来的字符串中的每一个空格都作为一个字符，是字符串中的一部分，不能任意变。

COBOL允许一行内写几个语句(语句间应以一个或多个空格分隔)，也允许一个语句写在多行上。

1.5.2 终端格式

为了方便用户书写和输入源程序，有的计算机系统允许使用终端格式。终端格式不设标号区，把续行标志区和A区结合在一起，除以下说明之外，它和ANSI格式使用规则一样。

(1) 终端格式的每一行可用到256列，即源程序每行最大长度为256个字符，在输入字符占满屏幕上一行(80列)后不要按回车键，继续输入，直到输入完本行(最多256个字符)全部字符为止。

(2) A区占用1～4列。第一列可以作为续行标志区。当第一列上写上连接符“－”，则表示本行是上一行的续行，而当第一列写“＊”，表示此行是注释行。需要说明的是，当第一列作为续行标志区时，A区将占用2～5列，而B区相应为第6～256列。

(3) B区占用第5～256列。

注意虽然编译系统可以处理每行256个字符，但是在列源程序清单时只能列出前125个字符，而用“·”代替未列出的字符。

§1.6 COBOL字符和COBOL字

1.6.1 COBOL字符

每一种计算机系统都规定了所允许使用的字符。并非所用的字符都能在COBOL程序中使用。例如在COBOL中，就不能用数学上的乘号“×”和除号“÷”，而要用“＊”和“/”代替。

有两种字符集：系统字符集和 COBOL 字符集。所谓系统字符集指的是在输入输出操作中允许出现的字符的集合。例如，有的计算机系统采用 ASCII 字符集，可以输出 ASCII 字符集中的多个字符。

COBOL 字符集指的是：在 COBOL 程序中允许出现的字符（用引号括起来的字符串中的字符除外）。每一种语言都规定了它自己的字符集，例如，BASIC 字符集、FORTRAN 字符集、PASCAL 字符集、C 字符集等，它们各不相同。例如在 BASIC 程序中要用到字符“#”、“!”等，而在 COBOL 程序中却不使用这些字符（用引号括起来的字符串中使用的字符不在此例）。每一种语言的字符集都比“系统字符集”所包含的字符少。它是源程序中所用到的字符的最小集合。

COBOL 字符集包括以下字符。

数字：0～9

大写字母：A～Z

小写字母：a～z（旧版本 COBOL 字符集不包括小写字母）

专用字符共 15 个：

+	加号
−	减号或连接号
*	乘号或星号
/	除号
=	等号
,	逗号
.	句号或小数点
;	分号
'	引号（有的用双引号，注意不分起始引号和终止引号，都用同一个字符）
(	左括号
)	右括号
<	小于号
>	大于号
	空格（即空白）。在本书中有时为说明某位置上有空格，以␣表示。
$	货币符号（美元符号）

在具体实现时，有的 COBOL 编译系统在上述 COBOL 字符集的基础上作少量调整，例如将美元符号“$”改成“元”的符号“¥”等。

“#”、“!”、“?”、“%”等字符属于系统字符集而不属于 COBOL 字符集，这些字符只能在 COBOL 程序中的字符串（用引号括起来）中出现，而不能在程序中其它地方出现。

1.6.2 COBOL 字

COBOL 字是由 COBOL 字符组成的，如同英文字母组成英文单词一样。COBOL 字是为了表示一定的意思，由字符组合而成的最小单位。如前面已见到过的，DIVISION，SECTION，MOVE，ADD…等都是 COBOL 字。

COBOL 规定，每一个 COBOL 字不允许超过 30 个字符。

COBOL 字分为两类：保留字和用户字（非保留字）。保留字指在 COBOL 中已规定作专门用途的字，它们代表特定的含义。例如 MOVE，在 COBOL 中它代表“传送”这一特定含义。同样，ADD 代表进行“加”的操作。这些保留字不能另作它用。附录Ⅵ中列出 COBOL 的全部保留字。

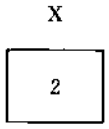
除了保留字以外，程序设计者还可以根据需要自己指定某些名字，例如程序名、文件名、节名、段名、数据项名等，本章例 1.2 中的 EXAM2 是程序名，A 和 B 是数据项名，这些都属于用户字。

在 COBOL 程序中，用用户名字代表一个对象，或称“标识”一个对象。如用一个文件名标识一个文件，用一个数据项名标识一个数据项，等等。因此，它们又被称为“标识符”(identifer)。通俗地说，标识符就是一个“名字”。

§1.7 数 据 名

1.7.1 数据名的概念

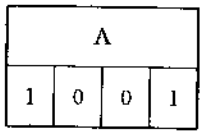
程序中常常要用到变量，例如 MOVE 2 TO X 中，X 就是一个变量，执行本语句后 X 的值是 2。在计算机中划出一定的内存单元来存放 X 的值。如：

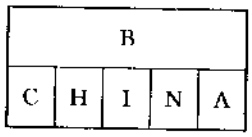


就如同一个匣子，里面可以存放一个数值一样，计算机中可以有许多这样的“匣子”，每一个都由程序设计者定名。上面这个例子，就是指定在内存中划出一定的单元(即一“匣子”)，在其中存放某一个值，这段内存单元定名为 X(即匣子的专用名)。正如我们设立一些保险箱，每个保险箱放在一定位置上，写上存户的名字，内放存户的存款。以后只要说出存户的名字，工作人员就能按照存户的名字找到该保险箱的位置，并把钱取出来。

因此，一个名字就代表内存中某段存储单元。只要指出这个名字，就能找到它所代表的内存单元，并可从其中调用或存放数据。在 COBOL 中，这个名字就叫做数据名(Data name)。因此，数据名可以认为是由程序设计者定义的某一片内存区的符号地址，即用一个符号代表一个地址。

数据名相当于其它语言中的变量名，它代表一个具体的数据项。COBOL 中指的数据是*广义的，即不仅指数值，也可以是字符。*





它表示在内存中某一段单元以 A 命名，它存放 1001 这个数值。在名为 B 的一段单元中，存放 CHINA 五个字符。注意，A 和 B 是程序中由程序设计者定义的名字，并非计算机内有一固定的区域名为 A 或 B。当一个程序运行完毕后，程序设计者所定义的所有数据名在计算机内就没有意义了。每一个程序都应根据需要确定所用的数据名，并规定它们在内存中占用区域的大小。

1.7.2 数据名的定名规则

(1) 每个数据名的长度为1～30个字符之间。

(2) 只能由字母、数字和连接符"-"组成，而且其中至少应有一个字母。连接符只能出现在数据名的中间，不能出现在数据名的两端(最前或最后)。如：

ABC,RATE-OF-PAY, 3DG4,A 1 2 3, 1234ABC,WANG-LI

都是合法的数据名。而：

123, JOHN.HENRY, WANG-,-A-B-C,

'ATE', ABC*DEF, 1981-8-31

都是不合法的数据名。

(3) 数据名中不能出现空格。如WANG LI不是一个数据名(而WANG-LI是一个数据名)。在COBOL中空格是一个非常重要的分界符，它表示一个COBOL字的结束，因此WANG被认为是一个数据名，LI被认作为另一个数据名。

(4) 不应选择COBOL的保留字作为用户定义的数据名。例如不能用MOVE,ADD,DIVISION,STOP等保留字来代表一个数据项。COBOL有三百多个保留字，要全记住它们是困难的。用连接符是避免误用保留字的好方法，如RATE-OF-PAY肯定不会是COBOL保留字，绝大部分保留字是不包含连接符的。此外，保留字中是没有用字母X开头的，也没有用数字开头的。用X和数字开头作数据名也可以避免误用保留字，但它又不大符合人们的习惯。

(5) COBOL并不要求数据名是有意义的英文字，A,B,C,XYZ均可作数据名。但由于COBOL的特点之一是"成文自明"，即看程序如同看一篇英文文章。因此在英语国家中的习惯是把数据名定为有一定含义的英文字，如用AMOUNT代表"金额"，用TOTAL代表"总和"等等，以反映出该数据项本身的含义。如：

```
IF AMOUNT > 100 GO TO BEGINNING.
ADD 1 TO TOTAL.
MOVE PART-NUMBER TO PART-NUM.
```

等。其中AMOUNT代表"金额"，当它>100时，就转向段名为BEGINNING(开始)的段。TOTAL为"累加总和"。PART-NUMBE代表"零件号"，PART-NUM是另一数据名，它的含义也是"零件号"，但在内存中它是另一个区域。这样看程序比较容易了解其含义。当然也可以不用上面这些作数据名，而用A,B,C,X,Y等代替。但看程序时要首先弄清楚这些A,B,C,X,Y等代表什么含义。我们不提倡用这种"代数符号"来作为数据名，它不符合"见名知义"的原则。也可以用汉语拼音，如用NIAN,YUE,RI代表"年"、"月"、"日"，用G-Z代替"工资"等。

§1.8 常　量

常量也就是通常所说的常数，本书中用"常量"而不用"常数"这一名称。因为COBOL中的常量不仅指数值常数，还包括非数值常量。有些书把它称之为"直接量"或"常字"。常量分三类。

1.8.1 数值常量(又称数值常数,数值常字。Numerical literal)

数值常量即数学上的常数,如+12300,451.67,-51.685 都是数值常量。

数值常量是由正负号,小数点,数字 0～9 所组成的字符序列,其中正负号和小数点可以没有,如 12,365 等。

注意:(1) 小数点不能多于一个,而且不能出现在常数的最右边,如 123. 和 123.45.1 写法是错误的。因为在 COBOL 中句点和空格表示一个句子的结束,因此,遇到 MOVE 25. TO I,就会把"25."认为一个句子已结束而发生混乱。

(2) 数值常量的长度不能超过 18 位数字。

(3) 至少要有一个数字。不能有多于一个符号。如+123.45-是不合法的。

(4) 数字之间不能有空格,如 15　25 并不代表 1525,它代表 15 和 25 二个量,空格是分界符。

1.8.2 非数值常量(又称非数值常数,非数值常字。Nonnumerical literal)

指用引号括起来的字符串。如'ABCD','$123.56','COBOL PROGRAM'都是非数值常量。例如我们想显示出 ABCD 这几个字符,用 DISPLAY 'ABCD'语句即可,把 ABCD 几个字符用引号括起来,注意引号本身不属于非数值常量本身,只是字符串的定界符号,在显示或打印时不出现引号。

可以将一非数值常量送给一数据项。如:

MOVE 'CHINA' TO COUNTRY-NAME.[注]

CHINA 要用引号括起来。把字符串 CHINA 送到数据项 COUNTRY-NAME 中。

非数值常量可以包括字母、数字、空格或其它字符。如果写 MOVE'123' TO A 表示将三个字符 123 送到 A 中,它仅代表三个孤立的字符 1,2 和 3,而无任何数值含义,不代表一百二十三。非数值常量不能参加运算,如'123'+'456'是错误的,而 123+456 是允许的。

如果想把引号也包括在非数值常量中,例如想显示出'ABCD'(包括引号),则可以写:

DISPLAY QUOTE 'ABCD' QUOTE

QUOTE 在英文中是"引号"的意思。它是一个 COBOL 保留字。在两个 QUOTE 之间的是一个非数值常量,显示的结果是'ABCD'共六个字符(包括两个引号)。即非数值常量的值为'ABCD'六个字符。

MOVE QUOTE 'CHINA' QUOTE TO A

表示将'CHINA'七个字符送到数据项 A 中。

注意 QUOTE 不能代替引号,如写 QUOTE ABC QUOTE 来代替'ABC'是错误的。也不能反过来,写成'QUOTE ABC QUOTE',这并不意味着一个非数值常量'ABC',而表示非数值常量 QUOTE ABC QUOTE,把引号内全部字符作为一个非数值常量,只有在引号外再使用 QUOTE,它才起字符串的定界作用。

非数值常量的长度,不应超过120个字符(有的计算机系统所用的COBOL放宽了这一

[注]:在 COBOL 程序中的引号可以是单引号或双引号,由具体的计算机系统规定。在计算机设备中是不分起始引号(')和终止引号(')的,都用同一个字符(')表示。

限制，可以多于 120 个字符）。

1.8.3 表意常量（又称字义常量，象征常数或赋形常数。Figurative Constant）

它用某些英文字（保留字）来代表某些特定的常值。如用 ZERO 代表数值 0，用 SPACE 代表空格。从这些字的意思可以知道它们所代表的是什么值，因此叫表意常量或字义常量。

MOVE ZERO TO A
MOVE 0 TO A } 二者等价

表 1.1 表意常量所代表的值

表意常量	所代表的值
ZERO ZEROS ZEROES	表示一个或多个零字符
SPACE SPACES	表示一个或多个空格字符
HIGH-VALUE HIGH-VALUES	表示一个或多个具有"最高值"的字符（每个字符的二进制表示为 11111111）。
LOW-VALUE LOW VALUES	表示一个或多个具有"最小值"的字符（每个字符的二进制表示为 00000000）。
QUOTE QUOTES	表示一个或多个引号字符
ALL 常量	表示由一个或多个该常量组成的字符串

表意常量可以出现在程序中使用数值常量或非数值常量的地方。读者仔细阅读表 1.2 所列举的用法举例，就可清楚表意常量的作用。

表 1.2

指令	假定已定义 A 在内存中的字节数	执行 MOVE 指令后 A 中的值
MOVE ZERO TO A	4	0000 （每个字节中均存放字符 0）
MOVE ZERO TO A	1	0
MOVE ZEROS（或 ZEROES）TO A	4	0000
MOVE SPACE TO A	4	␣␣␣␣（空格）
MOVE SPACE TO A	1	␣
MOVE HIGH-VALUE TO A	4	在 4 个字节中全部二进位上值为 1，即每个字节中均为二进制的 11111111
MOVE LOW VALUE TO A	4	每一字节中均为二进制的 00000000
MOVE QUOTE TO A	4	''''
MOVE ALL '*' TO A	4	* * * *
MOVE ALL 'AB' TO A	4	ABAB
MOVE ALL 'ABC' TO A	5	ABCAB

说明：(1) 表意常量的单数形式和复数形式是等价的，如 ZERO 和 ZEROS，ZEROES 完全等价。

(2) 如将一个表意常量传送给一个数据项，则表意常量的字符长度由该数据项长度决定。如：

```
MOVE ZERO TO A
```

A 如占 3 个字节，则表意常量 ZERO 的长度亦为 3 个字符，传送后数据项 A 中三个字节中均为 0。A 如占 2 个字节，则两个字节中均为 0。

(3) 如表意常量不和数据项发生联系，则认为此表意常量的字符长度为 1，如：

```
DISPLAY ZERO
```

在执行此语句时，显示(打印)出一个零字符。

§ 1.9 COBOL 所处理的数据的特点

1.9.1 层次的概念

在数据处理中，许多数据并不是互相孤立的。它们之间存在一定的联系。譬如，在统计工资中，某一个职工的工资清单中包含若干项：职工名、收入、扣除、实发工资等。它们是互相联系的，如果说收入 169 元，而不说明是哪个职工的，那是没有意义的。我们可以把属于同一职工的各数据项组织在一起，以一个总的数据名"工资统计"(GZTJ)来代表它。见图 1.7。在本例中数据名是用汉语拼音来表示的。

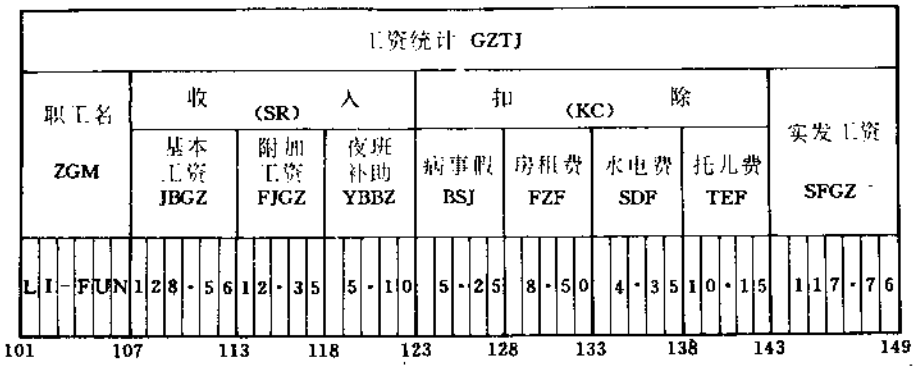


图 1.7

我们把某一职工的各数据组成一个整体 GZTJ 存放在内存中，这些数据在内存中是连续存放的。假设从内存地址 101 开始存放，职工名(ZGM)占 6 个字节，内放某一职工名字 *LIFUN，接着是基本工资(JBGZ)，占 6 个字节，即 107～112 字节，内放值 128.56，附加工资* (FJGZ)占 5 个字节，在 113～117 字节，值为 12.35，……依此类推，直到该职工的最后一项——实发工资(SFGZ)，内存地址为 143～149。如果我们写 MOVE GZTJ TO X，则将 101～149 字节中的全部内容一起取出来传送到 X 中。如果只写 MOVE ZGM TO X，则只将 101～106 字节中的内容(LI-FUN)取出来送到 X 中。

可以看出，数据间存在着从属关系，例如工资统计(GZTJ)这一总项下包括职工名

(ZGM)、收入(SR)、扣除(KC)、实发工资(SFGZ)几项，而收入(SR)又包括基本工资(JBGZ)、附加工资(FJGZ)、夜班补助(YBBZ)等几项。同样扣除(KC)又包括几项。我们把这种关系叫层次结构。如同一棵倒立的树。见图 1.8。

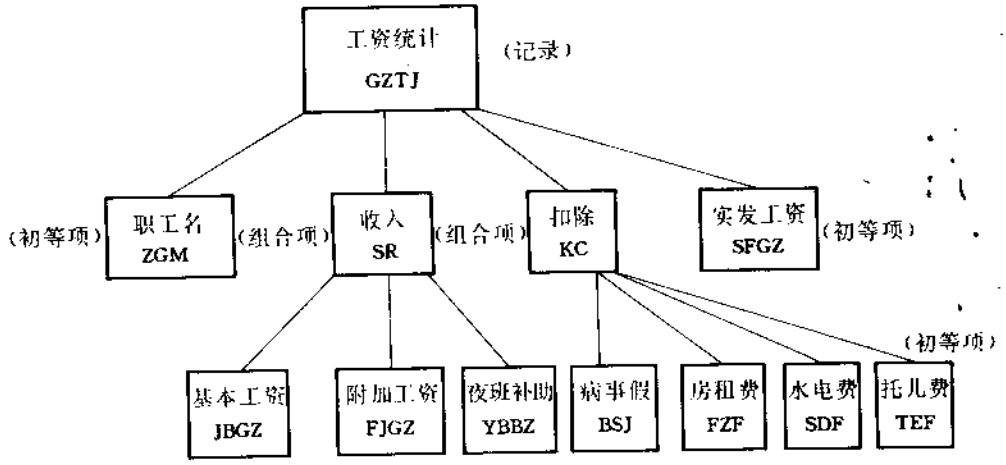


图 1.8

我们把工资统计这一总的数据项称为一个记录(record)，它包括一个职工的工资统计中的各有关数据项。记录下分为几项，其中职工名(ZGM)和实发工资(SFGZ)不能再分了，称为初等项或基本项(Elemention item)。初等数据项是数据的基本单位。而收入(SR)和扣除(KC)还可分为若干项，因此它们是组合项(Group item)。基本工资(JBGZ)、附加工资(FJGZ)、夜班补助(YBBZ)，病事假(BSJ)、房租费(FZF)、水电费(SDF)、托儿费(TEF)等是初等项。

为了反映某数据在一个层次结构中的层次，我们引进层号的概念，层号用二位整数来表示。用不同的层号表示它们之间的从属关系。我们把上述"工资统计"所含的各数据项在数据部中用以下不同层次来表示。层号小的表示高的层次。

```
01      GZTJ.          (工资统计)
  02    ZGM...         (职工名)
02      SR.            (收入)
  04    JBGZ...        (基本工资)
  04    FJGZ...        (附加工资)
  04    YBBZ...        (夜班补助)
02      KC.            (扣除)
  04    BSJ...         (病事假)
  04    FZF...         (房租费)
  04    SDF...         (水电费)
  04    TEF...         (托儿费)
02      SFGZ...        (实发工资)
```

关于层次和层号的概念在第四章中还要详细介绍。

1.9.2 记录和文件的概念

记录(record)是具有一定层次关系的一组数据项的最大集合。

它是内存中具有独立逻辑含义的最大可存取项，具有最高的层次，即层号为01。例如在工资统计中，内存中最大的存取单位是一个记录，即GZTJ。每一个职工的工资统计分别为一个记录。

以下是记录的例子：

学生成绩记录：包括学号、姓名、性别、各门课成绩、总平均成绩、升留级状况。

产品记录：包括产品名称、产品编号、单价、等级、库存量。

销售记录：包括商品名、商品号、单价、销售数、库存数。

工人出勤记录：包括职工号、姓名、本月出勤天数、病假天数、事假天数、旷工天数、应扣工资。

银行存户记录：包括存户名、原存款额、当天存(取)金额、结余金额、利率。

多个记录可以组成一个文件(file)。例如，一个学生成绩统计是一个记录，而一个年级或一个学校中许多学生的成绩统计就是一个文件——学生成绩文件。同样，许多个工人出勤记录组成全厂出勤统计文件，许多个销售记录组成全公司的销售文件等等。一个文件中包含多少个记录并无规定，根据实际情况的需要安排。

应当指出，文件是被记录在外部介质(如卡片、纸带、磁盘、磁带等)上的。例如，每一个学生的成绩统计记录穿在一张卡片上，全校若干个学生的成绩就需要由一批(若干张)卡片。这就是记录在卡片上的学生成绩文件，或者称卡片文件。统计时，将卡片上的信息一张一张地输入到计算机中，由计算机分别对每一个学生的成绩记录进行统计处理。也可以将若干个职工工资记录存放在一个磁盘文件中，需用时逐个记录调出进行处理。注意：不要误认为"在内存中许多个记录组成一个文件，文件占据一大片内存区"。在COBOL处理的问题中，在内存是找不到存储一个文件的存储区的。内存中最大的存取项是记录而不是文件。

可以说，文件是建立在外部介质上的记录的集合。任何一种外部设备的介质都可以作为文件的载体。例如，一条穿了孔的纸带，就是纸带文件。每运行一个程序从打印机输出的信息，就是一个打印文件，而其中每一个打印行就是一个记录。从读卡机读入的每张卡片记载一个记录，而一叠卡片就是一个卡片文件。文件也可以存放在磁带或磁盘上，这就是磁带文件或磁盘文件。

要给每一个文件起一个名字(文件名)以便与其它文件相区别。例如，如果我们把一个磁盘文件定名为A，则在COBOL程序中指定"从文件A中读入一个记录"，就是指通过磁盘机将一个记录上的数据输入到计算机中去。

在COBOL程序中对数据的存取是以文件为对象，以记录为单位的。文件、记录、数据项的关系可以用图1.9表示。

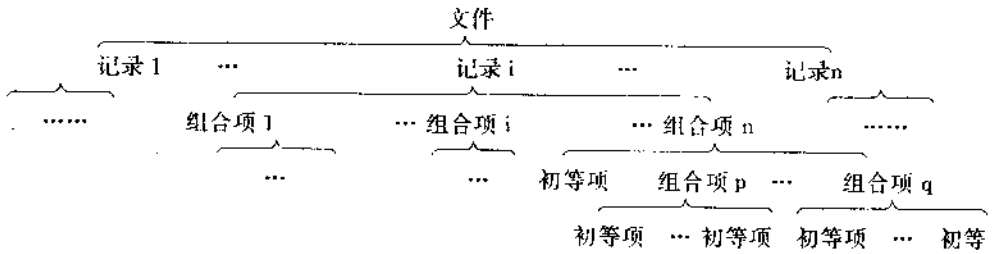


图 1.9

可以把若干个数据文件组织在一个"库"中，这就是数据库(Data library)或称"文件

库”。一个数据库包含若干个数据文件，例如，由职工出勤文件、职工工资文件、职工生产情况文件组成一个职工情况的数据库(见图 1.10)，可以调用不同文件中的内容。

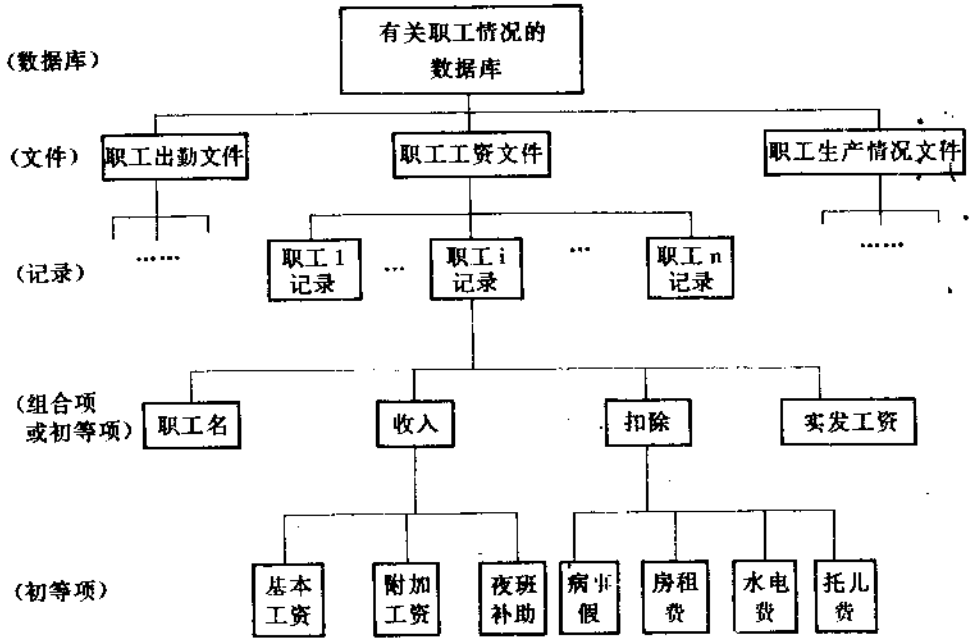


图 1.10

习　题

1.1　数据处理的问题有什么特点？

1.2　COBOL 源程序包括哪几个部分？它们的作用是什么？

1.3　什么叫数据名？下面这些是否是合法的 COBOL 数据名？

WANG，TAN-1，CHANG-SAN，＊123，1245，OHINA，END-OF PROGRAM，(A+B)/C，LI HAO，GO TO，STOP

1.4　常量分几类？说明下面的常量分别属于哪一类？哪些为不合法的写法？

125.87，－187.06，.123，126.，$66.8，‘876.5’，‘PEOPLE’，PAY，SPACE，77,000.6，ZERO，‘0’，QUOTE

1.5　什么叫数据组织的层次？举一实际生活中的例子，写成层次形式。

1.6　什么是初等项？什么是组合项？组合项能否包含组合项？

第二章　过程部初步
——最基本的过程部语句

§2.1 引　　言

前已指出，过程部是COBOL 程序中的核心部分，它决定计算机应进行什么操作。因此，学习COBOL 时，不应该把四个部分并列地、孤立地进行学习。而应当以过程部为主体，找出几部分之间的有机联系。这样才能把 COBOL 学活。有的初学者常常感到 COBOL 枯燥难学。原因之一就是一开始就陷入到较难理解的数据部中。而此时他们还不知道这些数据描述是做什么用的。针对这种情况，本书使读者首先接触过程部。由于过程部语句比较容易理解，同时有了过程部的概念后再接触其它部分，对其它部分的作用也就比较容易理解了，这样做可能会减少初学者学习 COBOL 的困难。我们在这一章中将介绍一些简单的 COBOL 程序，使读者逐步了解程序中几部分的相互联系。然后在以后各章中对其它三部分作详细介绍。根据我们的实践，按这样的顺序学习，对初学者可能是适宜的。

下面，先介绍过程部的几个特点：

（一）过程部是程序中的第四部分，它以部头 PROCEDURE DIVISION 开头。部头从 A 区（第 8～11 列）开始书写。在第一章中已知：过程部下面分若干节，节下面分段，段由若干句子组成，句子以句号"."和空格结束。句子由若干语句组成，语句间可用分号和空格或用空格分隔。如：

```
PROCEDURE DIVISION.          (过程部头)
A SECTION.                   (A 节)
A1. MOVE 1 TO I              (A1 段)
    MOVE 2 TO J.
A2. ADD 1 TO J.              (A2 段)
B   SECTION.                 (B 节)
⋮
```

本例的过程部下有两节：A 节和 B 节。在 A 节中有两段：A1 段和 A2 段。A1 段中只包含一个句子（句子是以句号和空格结束的，在 A1 段中只有第二个语句 MOVE 2 TO J 后才出现句点）。此句子包含二个语句：MOVE 1 TO I 和 MOVE 2 TO J。A2 段中只有一个句子，此句子中只包含一个语句：ADD I TO J。

在简单的程序中，过程部下一般不设节，而直接由段组成。有的 COBOL 系统甚至允许可以不设段，直接由句子组成。如：

```
PROCEDURE DIVISION
DISPLAY 'HOW DO YOU DO'
STOP RUN.
```

整个过程部只包含一个句子。

（二）过程部的语句都以一个动词（verb）开始，如 MOVE，OPEN，READ，WRITE，IF（IF 在 COBOL 中也作为动词对待）。它表示计算机应执行的操作。如：

```
MOVE 100 TO X.
ADD Y TO Z.
IF X>100 DISPLAY X.
```

这种写法近于英语中的句子，比较好懂。

（三）语句中的动词后面一般要跟以一个操作的对象。操作对象可以是数据名或文件名。如：

```
MOVE X TO Y.
```

表示将 X 的值送到 Y 去。操作的对象是数据项，它们是内存中某一段单元中的内容。

READ，OPEN，CLOSE 等语句的操作对象是外部文件，在这些动词后跟的是文件名，表示对与文件名相联系的外部文件进行某种操作。如 OPEN ABC，CLOSE ABC，READ ABC 等。ABC 是文件名。

（四）过程部的语句一律从 B 区开始书写，即从 12 列以后开始书写。一个语句可以任意写在一行或几行上。续行也应从 B 区开始。

§2.2 输入输出语句

2.2.1 接收语句(ACCEPT 语句)

在数据处理中，要处理的是大批数据，一般都是在外部介质（如磁盘、磁带）上建立数据文件（例如由若干工资记录组成的全厂工人工资文件）。如果需要用到这个文件中的数据，就要通过 READ 语句进行读入操作。被读的文件称为“输入文件”。在程序中用到的所有文件都需要在环境部中指定程序中用到的文件名与实际外部文件（如外部设备）的联系。在数据部也要对文件加以描述。指定数据结构和各数据项所占的内存单元长度以及数据形式。

如果只要输入少量的数据，则不必定义文件，COBOL 允许使用 ACCEPT 语句直接从终端键盘或系统指定的输入设备上输入少量的数据，使用起来比较方便。我们已经在前面几个例子中遇到过 ACCEPT 语句了。现在对它作必要的说明。

（一）ACCEPT 语句的一般格式

ACCEPT 标识符 [FROM 助忆名]

对这种“一般格式”说明如下：

（1）这里标识符(identifier)指的是能唯一地标识一个数据项的数据名，不能唯一地标识一个数据项的数据名不是标识符。有时在同一个 COBOL 程序中出现二个以上相同数据名，譬如在不同的组合项中可以包含相同名字的数据项。如

```
01 A.
   02 B1.
      03 C…
      03 D…
   02 B2.
      03 C…
      03 D…
```

在组合项 B1 和 B2 中，分别包含 C 和 D。即 C 和 D 数据名不是唯一的，它们不能称为标识符。不能写成：ADD C TO D。因为不能区分到底是哪一个 C，哪一个 D。而必须写成：

```
ADD C OF B1 TO D OF B1.
```

或 ADD C OF B2 TO D OF B2.

即说明是 B1 中的 C 和 D，或 B2 中的 C 和 D。关于这种数据名的限定的用法将在第 9 章中介绍。C OF B1，D OF B1 等是唯一地代表一个数据项，因此是标识符。

(2) 对这种一般格式，我们按 COBOL 的约定来理解它们。请看本书附录Ⅱ：关于 COBOL 语言格式的说明。在该附录中对各种语法符号的含义作了详细说明。例如：花括号的作用是表示在花括号中的各项中选择其一。方括号则表示其中各项是可选项(即可以有也可以无此项内容)。如果在方括号或花括号后面跟有省略号…，表示它前面方括号或花括弧中的内容可以重复若干次。有下划线的保留字表示这个保留字是在语句中必须写(不可省略)的，而没有下划线的保留字在语句中是可以省写的。

(3) 请注意不要把一般格式和实际的写法混淆起来。在实际语句中不能出现省略号，同样也不能出现方括号，花括号等。它们仅仅是说明怎样使用这些语句格式。在使用时应把它们变成有实际内容的语句。例如程序中出现下面的语句显然是不对的。

ACCEPT A [FROM B]

在 ACCEPT 语句的一般格式中，如不写 FROM 部分，如 ACCEPT A，则表示从系统隐含指定的设备上读入一个数据给 A。每种计算机系统都分别规定隐含的输入设备。一般指定为控制台(微机系统则指定为终端键盘)，有的中型机系统则指定为软盘输入机等。

如果不想从系统隐含指定的设备上输入，想另指定一种设备(譬如系统隐含规定的输入设备是键盘，而今想从软盘输入机上接收数据)，可以用"FROM 助忆名"可选项。所谓"助忆名"(又译"记忆名")是一个用来代表一个外部设备的名字。由于有的计算机系统中的外部设备名字(逻辑设备名)不好记，所以允许用户自己用一个名字来代替它，以帮助记忆，因此称"助忆名"，即设备助忆名。需要在环境部中事先说明"助忆名"和哪种外部设备相联系。如：

```
ENVIRONMENT DIVISION.          (环境部)
CONFIGURATION SECTION.         (配置节)
SPECIAL NAMES.                 (专用名段)
CONSOLE IS ABC.
```

在不同计算机系统中，每一种外部设备都由系统给它规定了专门的逻辑设备名(即系统给该设备指定的专用名字，请查阅所用的计算机系统的手册)。如果某计算机系统 COBOL 中以 CONSOLE 代表控制台，在 COBOL 程序的环境部中的专用名段内写：CONSOLE IS ABC，就表示在本程序中以 ABC 名字来代表控制台。这样，如果在过程部中有：

ACCEPT T FROM ABC.

就表示要从控制台接收数据。ABC 就是助忆名。

(二) ACCEPT 后面只能跟一个标识符，不能在一个 ACCEPT 语句中写两个以上标识符。如 ACCEPT A，B 是错误的。

但可以用组合项。如果在数据部已定义 A 为组合项，包含 A1，A2，A3。

A					
A1		A2		A3	

则可以用 ACCEPT A。此时一次应输入 6 个字节的内容。也可以写成：

ACCEPT A1

```
ACCEPT A2
ACCEPT A3
```

此时数据应分三行敲入，每行以'回车'结束。

（三）用 ACCEPT 语句从键盘上接收数据，需要人工干予计算机的执行，因此影响计算机的效率，只有输入少量数据时才使用它。有时希望从键盘上输入某些信息以检查运行情况（特别在调试程序时），用 ACCEPT 语句比较方便。

2.2.2 显示语句(DISPLAY 语句)

将少量数据从计算机内存中输出到某一指定的外设上，可用 DISPLAY 语句。它不必定义文件，只要在 DISPLAY 语句中直接写出数据名即可。

例如：

```
DISPLAY T.                          (显示出 T 的内容)
DISPLAY T1,T2,T3,T4.                (显示出 T1,T2,T3,T4 的内容)
DISPLAY 'TODAY IS MONDAY'.          (显示出一个字符串)
DISPLAY 'MY AGE IS', X.             (如果 X 中内容为"25",则打印出 MY AGE IS 25)。
```

（一）一般格式

```
DISPLAY { 标识符 1 } [ 标识符 2 ] …[UPON 助忆名]
        { 常  量 1 } [ 常  量 2 ]
```

以上几个例子都是无 UPON 可选项的。用 DISPLAY 语句可以显示出内存中某一初等数据项或组合项的内容，也可以显示出数值常数或非数值常量。

```
DISPLAY QUOTE 'MY NAME IS WANG HAO' QUOTE
```

则显示出'MY NAME IS WANG HAO'(包括引号)。

如果用 DISPLAY 语句显示一个表意常量，则输出一个字符。DISPLAY ZERO 打印出一个字符'0'。

（二）在什么设备上显示数据。如果没用 UPON 可选项，在计算机系统隐含指定的输出设备上显示数据。一般多用户分时系统和微机系统隐含指定的输出设备为终端显示器，有的则指定为宽行打印机。当执行 DISPLAY A 时，即在显示屏或宽行打印机上显示(或打印)出 A 的值，无需程序员另行指定。

但如果程序员想改变输出的设备，即不想从已隐含指定的设备上输出，而想改为在另一设备上输出，则可以使用"UPON 助忆名"可选项。在环境部的"专用名段"将欲输出的设备的逻辑设备名与程序员自己任意定的名字(助忆名)联系起来(方法和上节介绍 ACCEPT 语句中用的相同)。例如程序编制者已在环境部的专用名段中指定以助忆名 ABC 代表控制台，则执行，

```
DISPLAY A UPON ABC
```

时，即在控制台上显示出 A 的值。在写程序时，应先弄清楚所用的计算机系统中 ACCEPT 和 DISPLAY 隐含指定的接收或输出数据的设备是什么。

（三）每执行一个 DISPLAY 语句，总是从一个新行开始显示的，如：

DISPLAY A,B,C 三个值连续显示在一行上。如果内容多，一行显示不完，会自动换行再显示。而：

```
DISPLAY A
DISPLAY B
DISPLAY C
```

则将 A,B,C 三个值显示在三行上。

(四) 在运行正式的程序时，一般不用 ACCEPT 和 DISPLAY，以提高计算机效率，减少程序员的干预。但在教学工作中或初学者开始上机时，用 ACCEPT 和 DISPLAY 比较灵活方便，不需指定文件，可以直接输入一个数据给内存中某一数据项(初等项或组合项)，也可以直接将内存中任何一个数据项(初等项或组合项)输出，而不需通过文件和记录。

(五) 程序举例

【例 2.1】

```
IDENTIFICATION          DIVISION.     (标识部)
PROGRAM ID.             EXAM2-1.
ENVIRONMENT             DIVISION.     (环境部)
DATA                    DIVISION.     (数据部)
WORKING STORAGE         SECTION.      (工作单元节)
77      X       PIC     9(4).
77      A       PIC     9(4).
77      B       PIC     9(4).
77      C       PIC     9(4).
01    T.
02    T1        PIC     9(4).
02    T2        PIC     9(4).
02    T3        PIC     9(4).
02    T4        PIC     9(4).
PROCEDURE               DIVISION.     (过程部)
S.  ACCEPT    A.
    ACCEPT    B.
    ACCEPT    C.
    COMPUTE  X   = (A + B)/C
    MOVE     A TO T1.
    MOVE     B TO T2.
    MOVE     C TO T3.
    MOVE     X TO T4.
    DISPLAY    T.
    STOP    RUN.
```

我们在内存中设立了一个组合数据项，名为 T。它包括四个初等项 T1,T2,T3,T4。每一初等项定义为四个字节(四位整数)。在内存中数据存储的情况见图 2.1。

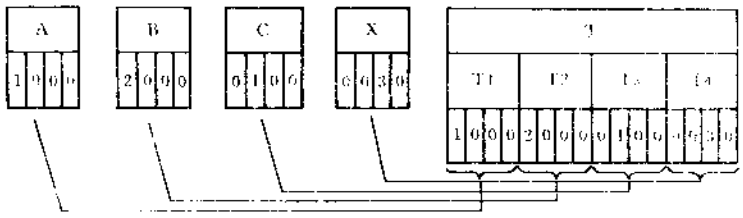


图 2.1

如果在所用的计算机系统中，ACCEPT 语句是隐含指定从终端键盘接收数据的，则在程序开始执行后，若我们想输入给 A、B、C 的数值分别为 1000，2000，100。则在键盘上输入形式为：

1000↙
2000↙
0100↙

它们分别送入相应的内存区。然后计算出 X 的值为 30。执行 MOVE 语句分别将 A，B，C，X 的值传送给 T1，T2，T3，T4。最后 DISPLAY 语句要求将 T 的内容输出，而 T 包括 T1，T2，T3，T4，则将这 16 个字节内容一起显示出来，显示结果为：

1000200001000030

共十六个字符，在一行上。

学到这里，读者可以开始模仿编一些简单的程序，并上机练习。用 ACCEPT 语句和 DISPLAY 语句来输入和输出数据。

2.2.3 读语句(READ 语句)

用 ACCEPT 语句只能从键盘输入少量数据，如果需要输入的数据比较多，用 ACCEPT 语句就很不方便，这时需要用 READ 语句。

COBOL 语言中数据的输入和输出主要是通过对外部文件的读和写进行的。执行读语句(READ 语句)就是从外部文件上读入数据输到程序中的数据项中。关于用 READ 语句从文件中读数据有以下几点说明：

(一) 什么是文件？在第一章中已经知道：所谓“文件”是记录的集合。一般所说的“文件”指的是外部文件，即建立在外部介质上的文件。例如，存在磁盘上的一组信息就是一个文件。人们常把一批互相有关的信息作为一组，存放在外部介质(如磁盘、磁带……)上，并给它起一个名字(称文件名)，用户想调用这组信息时，无须具体指出它在外部介质上的实际物理位置，只需指出它的名字即可，计算机就会根据文件的名字找到相应的文件，这就是说，文件是按名字存取的。

根据文件中信息的不同用途，可以分为程序文件(如源程序文件，目标程序文件等)和数据文件(它由数据记录组成，每个记录中存放一组数据)。执行读语句时，就从数据文件中读取数据。

外部设备也作为文件来处理，例如：打印机可看作打印文件；从读卡机输入一叠卡片上的数据，可认为是读一组卡片文件……。

每一个计算机的 COBOL 系统都规定了外部设备或外部文件的命名方法。例如在 IBM-PC 微机 COBOL 系统中，以“PRINTER”代表打印机，以“DISK”代表磁盘机，以“CONSOLE”代表控制台(或终端)。对磁盘上的每一文件亦各有文件名。例如 A:COBOL1.COB 表示在 A 盘(即当前在 A 驱动器中的磁盘)上名为 COBOL1.COB 的文件，其中“COBOL1”称为文件基本名，“COB”称为“扩展名”或“后缀”。如果想从某一文件中读数据，应在 READ 语句中指明文件名。

(二) READ 语句的最简单的格式为：

READ　文件名

例如：

```
READ IN-FILE
```

表示从 IN-FILE 文件中读入一组数据。但是 IN-FILE 代表哪一个外部文件呢？应该说明：在 READ 语句中并不直接使用外部文件实际的名字，上面的 IN-FILE 是程序设计者自己指定的一个名字，用它来代表一个实际存在的外部文件。

IN-FILE 称为“内部文件名”(即程序内部有效的文件名)，它与外部文件之间的对应关系是在环境部中指定的。在环境部中有以下部分：

```
ENVIRONMENT    DIVISION.        (环境部)
INPUT-OUTPUT   SECTION.         (输入输出节)
FILE-CONTROL.                   (文件控制段)
    SELECT IN-FILE ASSIGN TO 外部文件名.
```

可以看出，在环境部中已将内部文件 IN-FILE 与某一外部文件联系起来了。因此在程序的过程部中用 READ 从 IN-FILE 文件读数据就是从上面指定的某一外部文件中读数据。不同的计算机对外部文件的命名方法不相同。关于在环境部中如何定义文件，将在第三章中详细介绍。

COBOL 规定：一个内部文件名的长度不应超过 30 个字符(有的计算机系统要求文件名的长度还要短些)。并且只能由字母、数字和连接符组成，至少包含一个字母。

有人可能会问：为什么有了外部文件名之后还要再起一个内部文件名呢？能否在 READ 语句中直接写上外部文件名而取消内部文件名呢？COBOL 之所以规定在程序中使用内部文件名是为了程序的可移植性。因为不同的计算机系统的外部文件名的定名规则不同，如果将一个 COBOL 程序从甲机器移到乙机器上运行，如果程序中使用内部文件名的话，程序基本上可以不必改动，只需修改环境部中的 SELECT 子句一处即可(例如将上面提到的某一外部文件名改为所用计算机系统规定的外部文件名)。如果不用内部文件名的话，就势必将程序中所有文件名都要进行修改，工作量大且易出错，程序可移植性差。

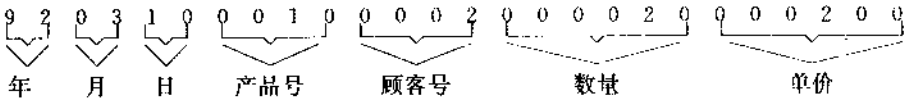
(三) 在 READ 语句中操作的对象是文件。前已叙及，COBOL 的存取操作是以文件为对象，以记录为单位的。每执行一次 READ 语句，就从指定的文件中读入一个记录(而不是整个文件)。这一点有些初学者往往容易搞混，误认为一次读入文件中全部记录。因此为避免误解，COBOL 语法规定在 READ 语句中可以写上一个关键字“RECORD”，例如：

```
READ IN FILE RECORD
```

其中 RECORD 表示“记录”，上面 READ 语句的含义是清楚的：读 IN-FILE 文件的一个记录。这个“RECORD”字是“可选项”，即它可写也可以不写，效果相同，写上 RECORD 无非使阅读程序者更清楚 READ 语句的含义，如果是熟练的程序人员，一般不会引起误解，完全可以不写“RECORD”，以使程序简练。

(四) 在计算机内存区中专门开辟一片存储单元用来存放从文件读入的信息。我们已知，从文件中一次读入一个记录，应当开辟的存储单元区的长度(字节数)应当等于文件中一个记录的长度。这个存储区称为“输入文件记录区”。每一个输入文件都有相应的“输入记录区”，例如在一个 COBOL 程序中用到 5 个输入文件，则开辟 5 个“输入记录区”，与之一一对应。

例如，我们从一个“产品销售文件”中读入数据，每一记录中包含的内容为：年、月、日、产品代号、顾客代号、数量、单价。设某一记录的具体数据如下：



我们希望将以上记录读入计算机内存，并将年、月、日、产品号、顾客号、数量、单价等数据分别送到各个相应的数据项中，以便以后利用它们进行计算或作其它处理。这就需要在"输入记录区"中划分出各个数据项(各占合适的长度)，以便存放相应的数据。定义输入记录区以及划分记录区中各数据项的工作是由数据部来完成的。

由于COBOL程序的几个部分之间有密切的关系，为了使读者对"读　语句"有比较清楚的了解，我们在此先把数据部定义记录区的方法作简单的介绍。针对上面输入数据的特点，我们可以在数据部中对记录区作如下描述：

```
DATA  DIVISION.          (数据部)
FD  IN-FILE  LABEL  RECORD  IS  STANDARD.
01  IN RECORD.  (定义记录区名为 IN RECORD.)
  02  DATE IN('日期'为一组合项)
    03  YEAR PIC 99.                    ('年',二位整数)
    03  MONTH PIC 99.                   ('月',二位整数)
    03  DAYY PIC 99.                    ('日',二位整数)
  02  PRODUCT-CODE PIC 9(4).            ('产品代码',四位整数)
  02  CUSTOMER CODE PIC 9(4)            ('顾客代码',四位整数)
  02  QUANTITY PIC 9(6).                ('数量',六位整数)
  02  UNIT-PRICE PIC 9(6).              ('单价',六位整数)
```

FD是File Description(文件描述)的缩写，表示从该行起是"文件描述体"。LABEL RECORD IS STANDARD意思是"标号记录是标准的"，所有磁盘文件和磁带文件都必须写明"标号记录是标准的"。01层定义"输入记录区"的名字为IN-RECORD，由于01层属于FD描述体，因此也就指定了记录区IN-RECORD和输入文件IN-FILE的关系。也就是从IN-FILE读入的数据存放在IN-RECORD记录区中。记录IN-RECORD包含五个02层，即划分五个数据项：DATE-IN，PRODUCT-CODE，CUSTOMER-CODE，QUANTITY，UNIT-PRICE。其中DATE-IN是组合项，又包括三个初等项YEAR，MONTH，DAYY(因为DAY是保留字，所以用DAYY代表'日')。规定它们的类型和在内存的记录区中占的字节数，如YEAR，MONTH，DAYY各占2个字节。PRODUCT-CODE，CUSTOMER-CODE各占4个字节，QUANTITY，UNIT-PRICE各占6个字节。它们都是数值型数据项。

这样，在执行一次READ语句后，磁盘文件中一个记录的数据便输入到内存记录区，按排列顺序分别送到记录区中各数据项中。图2.2表示文件中记录中的数据是如何送到各数据项的。在磁盘文件的各记录中的数据是不分什么数据项的，只是按一定顺序把数据存放在各字节中。由于在数据部中已定义了记录区IN-RECORD中各数据项的顺序和长度(第一个初等数据项为YEAR，占两个字节，第二个是MONTH，占两个字节，第三个是DAYY，占两个字节，后面一个数据项是PRODUCT-CODE，占4个字节，……。在将文件中第一个记录读入时，将第1和第2个字节的内容(92)送给数据项YEAR，将第3和第4个字节的内容(03)送给数据项MONTH，将第5和第6个字节的内容(10)送给数据项DAYY。将第7～10字节的内容送给数据项PRODUCT-CODE，……余类推。

也就是说，磁盘文件的记录中各数据排列次序和长度应和数据部对记录区中各数据项的描述一致。如果在数据部中对QUANTITY数据项改为5个字节，UNIT-PRICE改为7

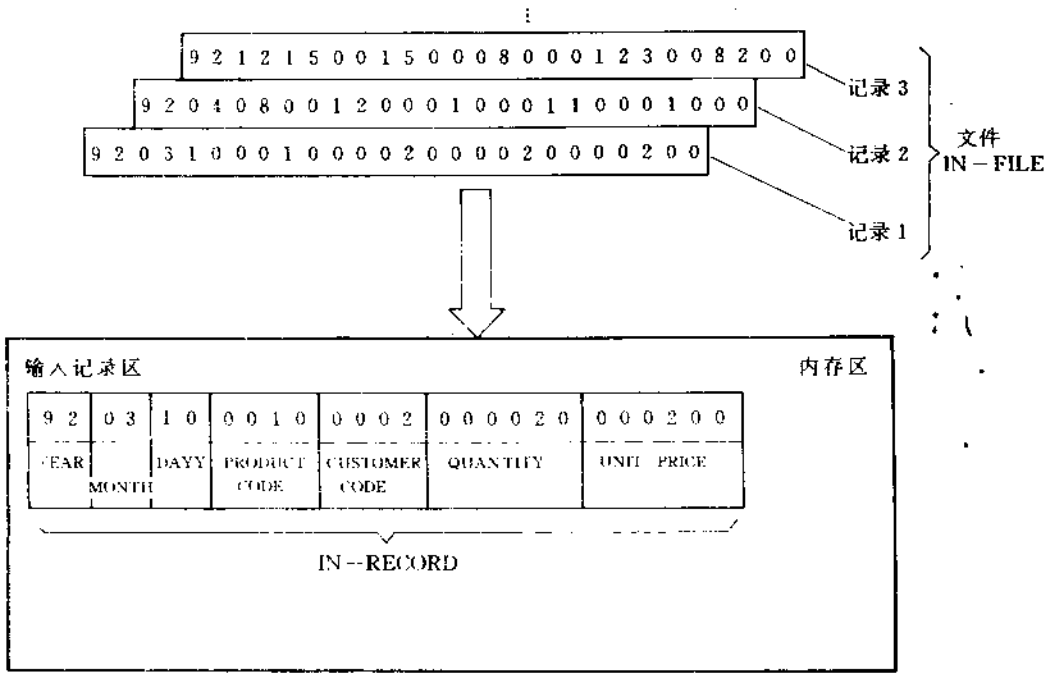


图 2.2

个字节，即：

02 QUANTITY PIC 9(5).
02 UNIT PRICE PCI 9(7).

而磁盘文件的记录中数据如前不变，则文件中的数据送入内存后，QUANTITY 的值不是 000020，而是 00002 了，UNIT-PRICE 的值多了一个前导零，为 0000200。见图 2.3 所示。

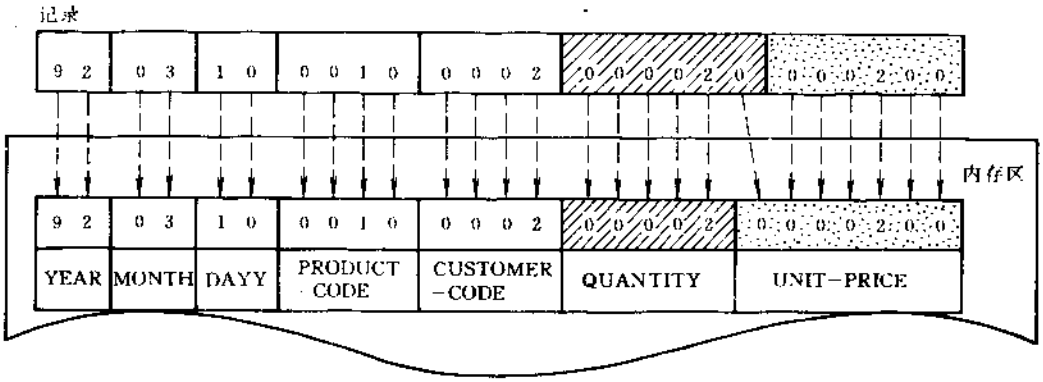


图 2.3

如果在读入一个记录后，接着再读入下一个记录，则第 2 个记录的数据仍然送入内存的同一输入记录区中。以后每读入一条记录，都是送到同一输入记录区中。即冲掉该记录区中前一组数据，而代以当前读入记录的内容。因此，应在读入文件的下一个记录以前，将前一记录处理完毕（如传送给其它数据项，或进行计算以得一新值，或存放到一个新文件中去等），

否则，内存区将不会保存前一记录的内容。

（五）文件读完时的处理。如一个文件包含若干个记录，为了能顺序读出文件中的各个记录，在文件中有一个指针，在开始时它指向第一个记录的开头（见图 2.4①）。在读完第一个记录后，该指针向后移动一个记录的长度，移到第 2 个记录的开头（见图 2.4②）。设文件有几个记录，在读完 n－1 个记录后，指针指向第 n 个记录的开头（见图中③）。在读完最后一个记录后，指针移到最后一个记录之后（见图中④）。此时就指向“文件尾”。文件尾有一个

图 2.4

“文件结束标志”。如果此时再执行一次 READ 语句，显然不可能正常读入一个记录而应当作特殊处理。当执行 READ 语句而遇“文件结束标志”时会给出一个“文件结束”的信息，允许程序规定作相应处理。这是由 READ 语句中的“AT END”子句来实现的。如：

```
READ IN-FILE AT END STOP RUN.
```

此语句的作用是：读一个 IN-FILE 文件的记录。如果该文件已读完而遇文件结束标志，则执行 AT END 子句中的语句 STOP RUN，程序结束运行。当然，也可以不写 STOP RUN 而写其它的操作语句，如 DISPLAY，MOVE，WRITE，…等语句，这是根据程序设计的需要而定的。

例如：

```
READING-DATA.      (段名)
    READ IN-FILE(文件名)
          AT END DISPLAY TOTAL STOP RUN.
    ADD QUANTITY TO TOTAL.
    GO TO READING-DATA.
        ⋮
```

这是一个循环，每读入一个记录后，即将其中的 QUANTITY 加到 TOTAL 中，直到读完文件的全部记录为止，此时结束运行。

（六）可以用一个 READ 语句读入一记录，并马上将记录区的内容转送到另一数据项中去。

```
READ   IN-FILE INTO T AT END STOP RUN.
```

这个读语句包括两个作用：(1)读入 IN-FILE 文件的一个记录到输入记录区。(2)将该记录区的内容传送到数据项 T 中去。它等价于下面两个语句：

```
READ IN-FILE AT END STOP RUN.
MOVE IN-RECORD TO T.
```

将记录 IN-RECORD（它是与文件 IN-FILE 相应的输入记录区）的内容全部传送到数据项 T 中，此时输入的数据同时存在于 IN-RECORD 和 T 两个域中。

（七）读语句的一般格式

READ 文件名 RECORD[INTO 标识符] [; AT END 执行语句]

可以看到：“INTO 标识符”和“AT END 执行语句”这两者是可选项。在关键字 READ，

INTO,END 下面没有下划线,它们是可以省略的。例如下面几种写法都是合法的:

```
READ IN-FILE RECORD INTO T; AT END STOP RUN
READ IN-FILE          INTO T; END STOP RUN
READ IN-FILE
READ IN-FILE INTO T
READ IN-FILE END STOP RUN
READ IN-FILE RECORD AT END STOP RUN
```

"AT END"前面的分号是可以省略的,它的作用只是用来与其前面的部分分隔开以便看起来清晰一些,不影响执行结果。

2.2.4 写语句(WRITE 语句)

(一) 将内存区中的数据输出到外部设备,主要是由 WRITE 语句来完成的。

与输入文件相仿,输出文件也在内存区开辟一个"输出文件记录区"(简称"输出记录区")。

将数据写到外部介质上,就形成输出文件(如打印文件磁盘文件)。对输出文件同样需要在环境部定义。也要用到内部文件名和外部文件名的概念。

注意:WRITE 语句中操作对象是记录,因此应写"记录名",而不是文件名,这是与 READ 语句不同的。用 WRITE 语句将一个输出记录区中的数据输出到相应的输出文件中去。例如:

```
WRITE T
```

表示将输出记录区 T 中的内容输出到与 T 相应的输出文件中去。记录区与输出文件的关系在环境部中确定(与输入文件的情况类似)。

用 WRITE 语句每次输出一个记录(而不能是记录中的一部分)。

(二) 输出设备的选择。在环境部确定所用的输出设备。如:

```
ENVIRONMENT DIVISION.      (环境部)
INPUT-OUTPUT SECTION.      (输入输出节)
FILE-CONTROL.              (文件控制段)
  SELECT OUTPUT-FILE ASSIGN TO 打印机名.
```

在实际写程序时,将"打印机名"代以所用计算机系统规定的打印机的专用名字。例如有的微型计算机系统规定用"PRINTER"作为打印机名,这时 SELECT 子句可写成:

```
SELECT OUTPUT-FILE ASSIGN TO PRINTER.
```

也可以将数据输出到磁盘上,形成磁盘文件。这时应在 ASSIGN TO 的后面写上磁盘文件名。

(三) 定义输出记录区。与定义输入记录区相仿,需要在数据部中对输出记录区和它下属的数据项进行定义。如:

```
DATA DIVISION.      (数据部)
FD OUTPUT-FILE LABEL RECORD IS OMITTED.
01 OUTPUT-RECORD.    (输出记录名为 OUTPUT-RECORD)
  02 DATE-OUT.
    03  YEAR              PIC  99.
    03  MONTH             PIC  99.
    03  DAYY              PIC  99.
  02  PRODUCT-CODE        PIC  9(4).
```

```
02  CUSTOMER-CODE      PIC  9(4).
02  QUANTITY          PIC  99.
02  UNIT-PRICE        PIC  999.
02  TOTAL             PIC  9(4)
```

上面第二行中"LABEL RECORD IS OMITTED"，表示"标号记录是省略的"。凡输出设备是打印机的，一律写成标号记录省略。

假设已将数据送到以上各数据项中，则执行"WRITE OUTPUT-RECORD"语句时，通过打印机输出该记录区中的全部数据(见图 2.5)。

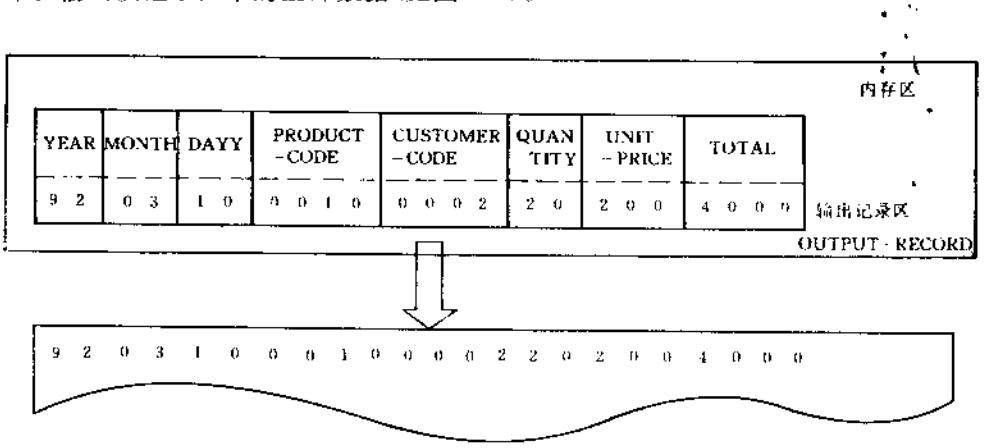


图 2.5

请注意记录与文件的联系。有的读者可能会奇怪：在 WRITE 语句中只写出了记录名 OUTPUT-RECORD，怎么在输出时就输出到打印机呢？请查一下数据部中，01 层定义了记录名 OUTPUT-RECORD，它是 FD 文件描述体的一部分，而文件名已定义为 OUTPUT-FILE，因此可知 OUTPUT-RECORD 是输出文件 OUTPUT-FILE 的相应的输出记录区。而在环境部中又已定义 OUTPUT-FILE 与打印机相联系。因此，将输出记录区 OUTPUT-RECORD 内容写出就必然是输出到打印机。

有的读者会认为这太麻烦了，何必绕这么一个圈子呢？其实，这正是 COBOL 的优点。在 WRITE 语句中不指出输出设备，只指出输出记录区，也就是在过程部中不牵涉到硬件设备。所有有关硬件设备的说明全部集中在环境部。如果想使数据从打印机输出改为向磁盘输出，只要修改环境部中的 SELECT 子句即可，过程部完全不必修改，在数据部中也只要将 FD 描述体中的"LABEL IS OMITTED"改为"LABEL IS STANDARD"。这就使改动输出文件时十分方便，不必对每个 WRITE 语句都作改动。有关环境部和数据部的规定和含义将在第三、四章中介绍，在此只是为了使读者对程序的几部分作用及其联系有初步的印象，先简单地作必要的介绍。

(四) 在用 WRITE 语句输出一个输出记录之前，应当向该记录区传送数据。在输出一个记录之后，该记录区的内容变成不确定的了，即不再保留原有内容，必须再对其传送新的内容，然后才可以再用 WRITE 语句输出。如：

```
MOVE T1 TO OUTPUT RECORD.
WRITE OUTPUT RECORD.
MOVE T2 TO OUTPUT-RECORD.
```

```
WRITE OUTPUT-RECORD.
```

第一个 WRITE 语句将 T1 的内容输出，第二个 WRITE 语句将 T2 的内容输出。

（五）纵向走纸的控制

（1）在打印机上输出时，往往需要用到不同的行间隔，譬如是一行接着一行打印，还是打完一行空一行再打第二行（隔行打印），甚至打印完若干行就跳到下一页的开头接着打印等等。COBOL 提供了行距控制的方便用法。先看一例：

```
MOVE T1 TO OUTPUT-RECORD.
WRITE OUTPUT-RECORD AFTER ADVANCING 2 LINES.
```

它的英文含义是清楚的："在前进两行之后打印 T 记录内容"（ADVANCING 和 LINES 两字可省略）。即先走纸二行（纸向上移动二行），然后输出 OUTPUT-RECORD 的内容。如果在执行此 WRITE 语句之前，打印机头的位置如图 2.6 中①所示（即停在第一行左端）。则在执行此 WRITE 语句时，先走纸二行，打印机头停在第三行左端（如图 2.6②所示），然后输出记录的内容（即 T1 内容）。在输出完此记录后，打印机头停留在该行最右边字符的后面（见图 2.6③）。

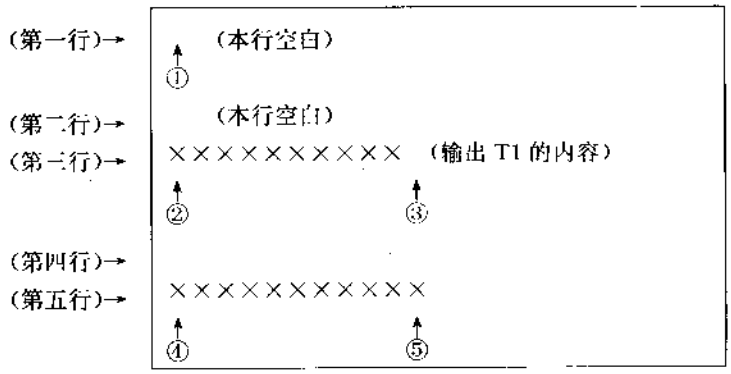


图 2.6

如果此时再执行下面语句：

```
MOVE T2 TO OUTPUT-RECORD.
WRITE OUTPUT-RECORD AFIER ADVANCING 2 LINES.
```

则又走纸二行，打印机头的位置在纸上第五行左端处（见图 2.6④），然后输出 OUTPUT-RECORD 记录的内容（亦即 T2 的内容）。执行完第二个 WRITE 语句后打印机头的位置如图 2.6 的⑤所示。

在 WRITE 语句中，除了可以用 AFTER 外，还可以用 BEFORE，如：

```
WRITE OUTPUT-RECORD BEFORE ADVANCING 2 LINES.
```

它的作用是"前进两行之前打印 OUTPUT-RECORD"，如果有以下两个写语句：

```
    ⋮
WRITE OUTPUT-RECORD BEFORE ADVANCING 2 LINES.
MOVE T1 TO OUTPUT-RECORD.
WRITE OUTPUT-RECORD BEFORE ADVANCING 2 LINES.
    ⋮
```

假设执行 WRITE 语句之前，打印头在图 2.7①位置，即在第一行的左端，该行原为空白，在执行第一个 WRITE OUTPUT-RECORD BEFORE ADVANCING 2 LINES 时，先将

OUTPUT-RECORD 记录输出，打印在第一行上。打印完本行后、打印头位置为②所示。然后“前进”两行，到③处，再执行第二个 WRITE OUTPUT-RECORD BEFORE ADVANCING 2 LINES 时再将 OUTPUT-RECORD 记录输出，也就是将 T1 的内容打印在第三行上，打印完本行打印头位置在④，然后再“前进”两行到⑤位置。这就是执行完以上两个 WRITE 语句后打印头的最后位置。

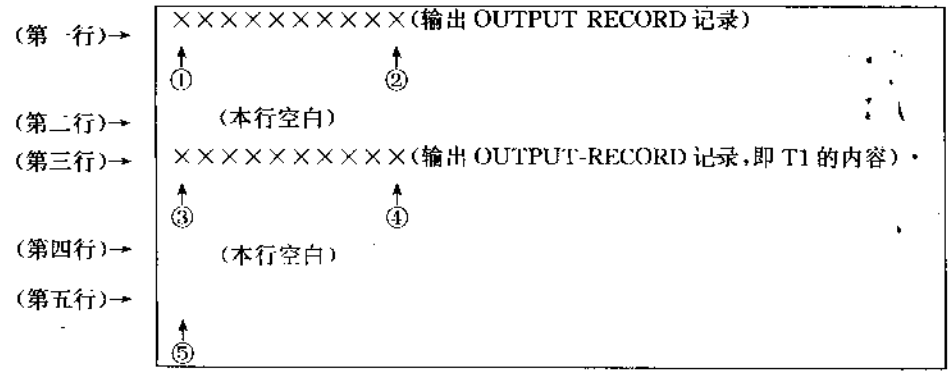


图 2.7

可以看出，用 AFTER ADVANCING 2 LINES 和 BEFORE ADVANCING 2 LINES，都是使两个打印行间空隔一行，但二者的不同在于用 AFTER 是“先移后打”，即先移动两行再打印，打印完后不再移行。而用 BEFORE 是“先打后移”，打印之前不移行，打完后再移行。

(2) 在同一程序中，如果第一个 WRITE 语句中用了 AFTER，则后面所有 WRITE 语句中最好也用 AFTER。如果第一个 WRITE 语句中用了 BEFORE，则后面最好全用 BEFORE。这是为了避免换行的混乱。例如：

```
MOVE T1 TO OUTPUT-RECORD.
WRITE OUTPUT-RECORD AFTER ADVANCING 2 LINES.
MOVE T2 TO OUTPUT-RECORD.
WRITE OUTPUT-RECORD BEFORE ADVANCING 2 LINES.
```

在执行第一个 WRITE 语句前，打印头在图 2.8 中①位置，执行完该 WRITE 语句后将 OUTPUT-RECORD 记录内容(即 T1 内容)打印在第三行上，然后打印头停在②位置。然后执行第二个 WRITE 语句，先输出后走纸，因此不换行而在第二行上接着打印输出的记录(即 T2 的内容)。最后打印头停在③处。显然这种输出格式并不是人们所希望的。

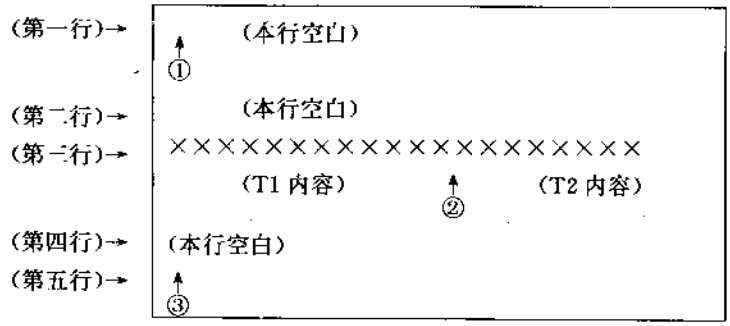


图 2.8

这样打印出来的结果肯定是不会令人满意的，读者可自己分析一下。

(3) 可以用数据名来代替整数用于 AFTER 和 BEFORE 之后，如：

```
WRITE OUTPUT-RECORD AFTER ADVANCING A LINES.
```

当 A 值为 3 时，相当于 ADVANCING 3 LINES.

(4) 可以用：

```
WRITE OUTPUT-RECORD AFTER ADVANCING PAGE.
```

表示先移到下一页的开头，再打印 OUTPUT-RECORD 记录，PAGE 是"换页"。

(5) 有时需要有一些特殊的控制，如要移到本页末或行叠印(不换行)，各 COBOL 编译系统为每一种特殊的功能分别规定了专用的名字。在环境部的专用名段中可以由用户自己定一个名字(助忆名)与此专用名相联系，譬如我们已在专用名段中指定助忆名 B 与"不换行(叠印)"相联系(计算机系统对"叠印"这一功能指定一个专用的名字，可查说明书)。则在过程部中写：

```
WRITE OUTPUT-RECORD AFTER B.
```

表示在上一次打印的行上不换行接着打印本行，即两行重叠。

使用这方面的控制，应查本系统说明书，了解专用名的规定和含义。初学者可不必深入学这部分，以后用时查一下即可。

(6) 在许多计算机系统的 COBOL 中，在用 WRITE 语句在打印机上打印记录时，输出记录区的第一个字符被系统作为"纵向走纸控制"之用。也就是说，输出的每一记录的第一个字符将不输出而作为控制走纸字符之用。因此如果我们对记录区 OUTPUT-RECORD 的描述为：

```
01 OUTPUT RECORD PIC X(133).
```

而 OUTPUT-RECORD 记录中内容为：

```
1234567
```

在输出记录区 OUTPUT-RECORD 中第一个字符是"1"，但打印时这个"1"用来控制纵向走纸，不输出。通俗地说，第一个字符被系统"吃掉"了。打印在纸上的是：

```
234567
```

少了一个字符。如果 OUTPUT-RECORD 记录内容改为：

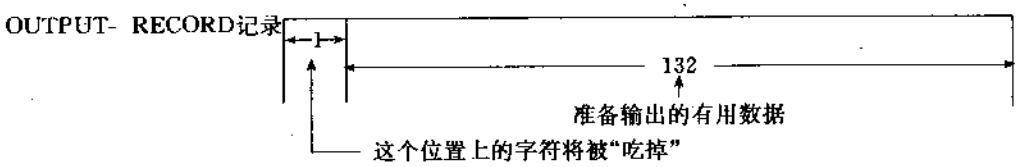
```
␣1234567
```

即加了一个空格在第一个字符处。执行 WRITE OUTPUT-RECORD AFTER 1 时，打印出：

```
1234567
```

因此，如果所用行式打印机的纸宽是 132 字符，则在写输出记录描述体时，应定义一个 133 个字符的区域。其中第一个字符供走纸控制用，后 132 个字符供用户作为输出信息区。如：

```
01  OUTPUT RECORD                (记录描述)
  02  FILLER PIC X.              (走纸控制位)
  02  OUT REC PIC X(132).        (输出信息区)
```



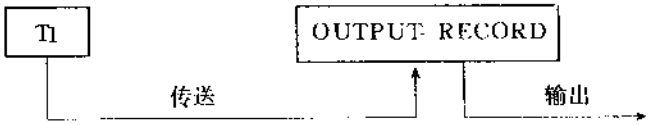
初学者应了解此规定，以免在输出时少打印一个字符而不知其所以然。但是也有的计算机系统所用的 COBOL 不"吃掉"第一个字符。使用 WRITE 语句时，应先了解本系统的规定。

(六) 可以用一个 WRITE 语句先把内存区中另一数据项的内容送给要输出的记录区，然后再输出。如：

```
WRITE OUTPUT-RECORD FROM T1 AFTER 3.
```

它相当于下面两个语句：

```
MOVE T1 TO OUTPUT-RECORD
WRITE OUTPUT-RECORD AFTER 3.
```



FROM 后面的数据项可以是组合项或初等项。如图 2.9。

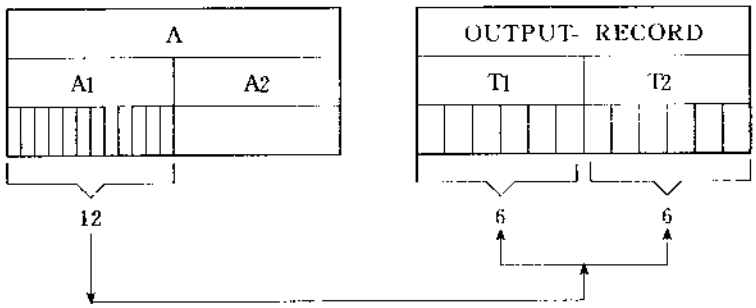


图 2.9

A1 是初等项，占 12 个字节。OUTPUT-RECORD 是记录名，下属 T1 和 T2，共占 12 个字节，当执行

```
WRITE OUTPUT-RECORD FROM A1 AFTER 1.
```

表示先将 A1 内容送到 T，按自左而右顺序放入 T1 和 T2，然后输出 OUTPUT-RECORD 记录。

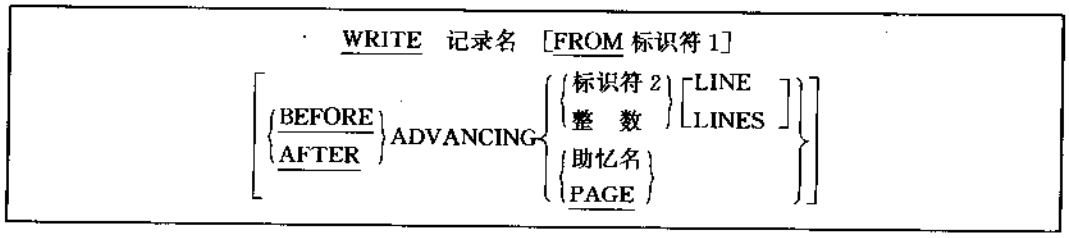
(七) 当不出现 BEFORE 或 AFTER 时，大多数系统按等价于 AFTER 1 处理。

如：

```
WRITE OUTPUT-RECORD AFTER 1.
WRITE OUTPUT-RECORD.
```

>等价

(八) WRITE 语句的一般格式：



2.2.5 打开语句(OPEN 语句)

(一) 程序中如果需要读文件或写文件,则该文件必须先用 OPEN 语句打开,系统在执行 READ(读)以前先检查该文件是否已在规定的外部设备上准备好。

```
OPEN INPUT X1, X2, X3.
```

表示打开输入文件 X1,X2,X3。以后可以用读语句从 X1,X2,X3 文件读入数据。

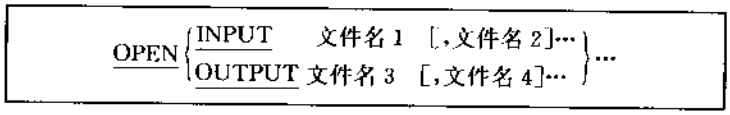
```
OPEN OUTPUT Y1, Y2, Y3.
```

表示打开输出文件 Y1,Y2,Y3。以后可以用写语句将相应的记录输出给这些文件。

(二) 一个 OPEN 语句可以打开一个或多个文件。也可以用一个 OPEN 语句同时打开若干个输入文件和输出文件。如:

```
OPEN INPUT X1, X2, X3
     OUTPUT Y1, Y2, Y3.
```

(三) OPEN 语句的一般格式:



2.2.6 关闭语句(CLOSE 语句)

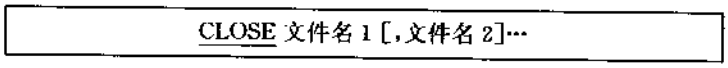
(一) 当对一个文件的读或写的操作已完成,就应关闭这不再使用的文件,使它不再涉入以后的数据操作之中。

如:

```
A. OPEN INPUT X-FILE
          OUTPUT Y-FILE.
B. READ X-FILE AT END GO TO C.
     ⋮
   WRITE Y-REC AFTER 1.
   GO TO B.
C. CLOSE X-FILE, Y-FILE.
   STOP RUN.
```

其中 Y-REC 是从属于文件 Y-FILE 的记录的名字。

(二) CLOSE 语句的一般格式:



(三) 注意:

(1) CLOSE 与 OPEN 用法不同，不必指出是 INPUT(输入)文件或 OUTPUT(输出)文件，只需指出文件名即可。

(2) 在一个程序中有 OPEN 必须也有 CLOSE，反之亦然，二者呼应。

对每一个在 OPEN 语句中打开的文件，必须在 CLOSE 语句中关闭。在一个 OPEN 语句中打开的几个文件，可以分别在几个 CLOSE 语句中关闭。反之，在几个 OPEN 语句中打开的文件，可以用一个 CLOSE 语句关闭。

(3) 文件关闭后，不能再对这些文件进行读写操作，如需再用，可以再打开。

§2.3 算术运算语句

2.3.1 加法语句(ADD 语句)

(一) 加法语句举例

先举具体的写法，然后介绍其规律。

```
ADD  A   TO  B              表示 A+B⇒B，即 A 的值加 B 的值，结果放在 B 中。
ADD  15  TO  C              表示  15+C⇒C
ADD  A,  B   TO  C              表示 A+B+C⇒C
ADD  15, 25  TO  C              表示 15+25+C⇒C
ADD  A,  B   GIVING  C          表示 A+B⇒C
ADD  15, 25  GIVING  T          表示 15+25⇒T
ADD  A, B  TO  C, D             表示 A+B+C⇒C
                                和 A+B+D⇒D
```

可以看出：加法语句有几种不同的形式。如：

(1) ADD A TO B。表示数据项 A 的值和数据项 B 的值相加，结果置于 B 中。如 A 原值为 10，B 为 100，此语句执行前后在内存中数据项的值变化如图 2.10(设 A 和 B 均已被定义为四个字节)：

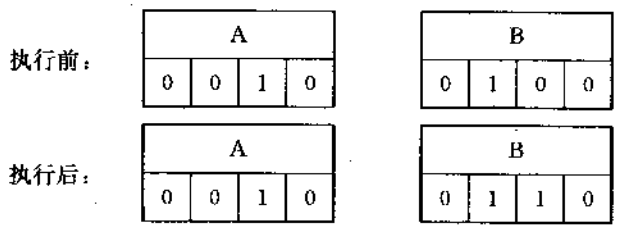


图 2.10

可以看到，A 的值不变，B 的值改变了。

ADD A,B TO C 表示将 A 和 B 都加在 C 上，即 A+B+C⇒C。A 和 B 之间可以有逗号也可以没有逗号，但空格是不能省的，用逗号来分隔各项只是为了看程序时清楚些，不影响程序的执行。

(2) ADD A,B GIVING C 表示将 A 和 B 相加，结果放在第三个数据项 C 中。执行前如图 2.11 所示。

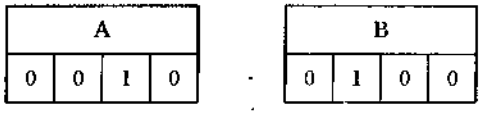


图 2.11

执行后如图 2.12 所示。

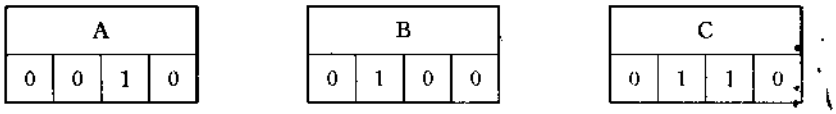


图 2.12

此时 A 和 B 的值都不变。相加的和送到 C 中。注意这两种用法的区别：

```
ADD  A, B  TO  C          (C 参加加法运算  A+B+C⇒C)
ADD  A, B  GIVING  C      (C 不参加运算  A+B⇒C)
```

(3) ADD A, B TO C, D 表示进行两个加法运算，A+B+C→C，A+B+D→D。即将 A 和 B 加在 C 上，同时，将 A 和 B 加到 D 上。

下面举例说明用加法语句进行运算时各有关数据项的值。设 A，X，Y，Z 均已分别定义，在执行 ADD 语句前 A，X，Y，Z 的值分别为 10，15，25 和 30，执行 ADD 语句后相应的 A，X，Y，Z 如表 2.1 所示。

表 2.1

执行后结果 / 数据项 / 执行的语句	A (原值 10)	X (原值 15)	Y (原值 25)	Z (原值 30)
ADD X TO Y	10	15	40	30
ADD A, X TO Y	10	15	50	30
ADD 5, Y GIVING Z	10	15	25	30
ADD A, X, Y GIVING Z	10	15	25	50
ADD A, 5, X TO Z	10	15	25	60
ADD A, X TO Y, Z	10	15	50	55
ADD 10, 50, TO A	70	15	25	30
ADD A, Z, Y TO X	10	80	25	30

(二) 用法注意：

(1) 在 TO 和 GIVING 后面只能跟数据名，而不能跟常量。如：

```
ADD  5  TO  5
ADD  A,  B  GIVING  100
```

是错误的。

(2) TO 前后的数据名的次序不要随便改换，如：

```
    ADD  A  TO  B        表示  A+B⇒B
而：ADD  B  TO  A        表示  A+B⇒A
```

二者含义不同。

TO 后面的是被加数，它的值是要改变的。

(3) GIVING 的后面可以跟几个数据名，如 ADD A，X GIVING Y，Z 表示 A+X⇒Y；A+X⇒Z。

(4) 参加运算的只能是数值量(数值常量、数值型的初等数据项或表意常量 ZERO)。它们值的长度不应超过 18 位数字(不包括数的符号和小数点)。

(三) 加法语句的一般格式

为了查阅使用方便，我们把 ADD 语句写成一般格式如下：

格式 1.

ADD { 标识符 1 / 常　量 1 } [,标识符 2 / ,常　量 2]…TO 标识符 m[,标识符 n]…

格式 2.

ADD { 标识符 1 / 常　量 1 } { 标识符 2 / 常　量 2 } [,标识符 3 / ,常　量 3]…GIVING 标识符 m[,标识符 n]…

下面是实际使用的加法语句例子：

```
ADD INTEREST, DEPOSIT TO BALANCE.
    (利息)      (存款)        (余额)
ADD REGULAR TIME, OVER TIME GIVING GROSS-TIME.
    (正常的工时)    (加班工时)        (总工时)
```

数据名 INTEREST，DEPOSIT…等是程序员定义的。用这些有含义的英文字是为了“成文自明”，不须再多解释便可以了解该语句的意思。

2.3.2 减法语句(SUBTRACT 语句)

(一) 减法语句举例：

① SUBTRACT B FROM A	从 A 中减去 B。A－B⇒A
② SUBTRACT B, C FROM A	从 A 中减去 B、C。A－B－C⇒A
③ SUBTRACT B, C FROM A, T	从 A 中减去 B 和 C。A－B－C⇒A 从 T 中减去 B 和 C。T－B－C⇒T
④ SUBTRACT B, C FROM A GIVING X	X－B－C⇒X

以上共有二种形式：①，②，③是从 FROM 后面的数据项中减去 FROM 前面的量，结果存放在 FROM 后面的数据项中，可以读为：“从 A 中减去 B”，“从 A 中减去 B 和 C”，“分别从 A 和 T 中减去 A 和 B”。从它的英文意思，这个语句的含义是很清楚的。④是另一种格式，它的意思是“从 A 中减去 B 和 C，把求出的差给 X”。它和①②③的不同在于 A 的值不变化，结果值送到另一数据项 X 中，而后者是在执行该语句时 A 的值将会改变。

注意：(1) GIVING 后面不能跟常量。如：

```
SUBTRACT A FROM B GIVING 10
```

显然是错误的。

(2) 如不带 GIVING 部分，即上面①、②、③形式，则 FROM 后面也不能跟常量。

```
SUBTRACT A FROM 10
```

同样是错误的。而下面的用法是正确的：

```
SUBTRACT 5 FROM 10 GIVING C
```

表 2.2 说明各数据项在执行减法语句前后它们的值的变化情况。设每一语句执行前，

X,Y,Z 的值分别是 100,70,30。

表 2.2

执行后结果 / 数据项 / 执行的语句	X (原值 100)	Y (原值 70)	Z (原值 30)
SUBTRACT Y FROM Z	100	70	−40
SUBTRACT Z FROM Y	100	40	30
SUBTRACT Y, Z FROM X	0	70	30
SUBTRACT 5, 10 FROM X	85	70	30
SUBTRACT Y, 20 FROM Z	100	70	−60
SUBTRACT Z FROM Y GIVING X	40	70	30
SUBTRACT Y, 15 FROM 100 GIVING Z	100	70	15

(二) 减法语句的一般格式：

格式 1

SUBTRACT {标识符 1 / 常 量 1} [,{标识符 2 / 常 量 2}]…FROM 标识符 m[,标识符 n]…

格式 2

SUBTRACT {标识符 1 / 常 量 1} [,{标识符 2 / 常 量 2}]…FROM {标识符 m / 常 量 m}
GIVING 标识符 n[,标识符 p]…

如果对加法语句的用法和一般格式清楚的话，对减法语句的用法和一般格式也是容易理解的。COBOL1974 允许在 GIVING 后跟多个标识符，如 SUBTRACT A FROM 100 GIVING X,Y,Z。其含义是将 100-A 所得的结果赋给数据项 X,Y,Z。

2.3.3 乘法语句(MULTIPLY 语句)

先看几例：

MULTIPLY A BY B	表示 A 被 B 乘，结果放在 B 中，A×B⇒B
MULTIPLY 0.5 BY B	0.5 被 B 乘，结果放在 B 中，0.5×B⇒B
MULTIPLY A BY B GIVING C	A×B⇒C
MULTIPLY 1.5 BY 3 GIVING C,A	1.5×3⇒C， 1.5×3⇒A
MULTIPLY A BY B,C	A×B⇒B，A×C⇒C

乘法语句也有两种格式。一种是不带 GIVING 部分的，如第一个语句，可读为“A 被 B 乘，结果放在 B 中”。第二种格式是带 GIVING 部分，如第三个语句，读为“A 被 B 乘，结果送到 C”。

与加、减语句类似，应注意：

(1) 当不带 GIVING 部分时，BY 后面不能是常量，只能是标识符(数据名)。如：

MULTIPLY A BY 0.5

是错误的。

(2) 带 GIVING 部分时，BY 后面可以是常数，而 GIVING 后面不能是常量。如：

MULTIPLY A BY 0.5 GIVING X

是合法的，而：

MULTIPLY A BY C GIVING 0.5

是错误的。

总之，存放值的项只能是数据名，不能是常量。

乘法语句执行前后各数据项的值变化举例见表 2.3。

表 2.3

执行后结果 / 数据项 / 执行的语句	A (原值 10)	B (原值 20)	C (原值 30)
MULTIPLY A BY B	10	200	30
MULTIPLY B BY A	200	20	30
MULTIPLY 0.5 BY A	5	20	30
MULTIPLY A BY B GIVING C	10	20	200
MULTIPLY A BY 0.5 GIVING B,C	10	5	5
MULTIPLY 1.5 BY 5 GIVING A,B,C	7.5	7.5	7.5

乘法语句的一般格式：

格式 1

$$\underline{\text{MULTIPLY}}\left\{\begin{matrix}\text{标识符 1}\\ \text{常　量 1}\end{matrix}\right\}\underline{\text{BY}}\ \text{标识符 2[,标识符 3]}\cdots$$

格式 2

$$\underline{\text{MULTIPLY}}\left\{\begin{matrix}\text{标识符 1}\\ \text{常　量 1}\end{matrix}\right\}\underline{\text{BY}}\left\{\begin{matrix}\text{标识符 2}\\ \text{常　量 2}\end{matrix}\right\}\underline{\text{GIVING}}\ \text{标识符 3[,标识符 4]}\cdots$$

2.3.4 除法语句(DIVIDE 语句)

先看几例：

DIVIDE A INTO B　　表示 $\frac{B}{A}\Rightarrow B$

即以 B 除以 A，结果送到 B。

DIVIDE A INTO B GIVING C　表示 $\frac{B}{A}\Rightarrow C$

即 B 除以 A，结果送到 C。

DIVIDE A BY B GIVING C　　表示 $\frac{A}{B}\Rightarrow C$

即 A 被 B 除，结果送到 C。

当除法语句中用 INTO 时，如 DIVIDE A INTO B，意思是 B 除以 A，即 $\frac{B}{A}$。而用 BY 时，

如:DIVIDE A BY B 是指 A 被 B 除,$\frac{A}{B}$。二者不要搞混。

和前面介绍的相似,当不带 GIVING 部分时,INTO 后面不能跟常量,如 DIVIDE A INTO 50 是错误的。如果带 GIVING 部分,GIVING 后面不能跟常量,如 DIVIDE A BY B GIVING 10 是错误的。

除法语句的一般格式:

格式 1

DIVIDE {标识符 1 / 常 量 1} INTO 标识符 2[,标识符 3]…

格式 2

DIVIDE {标识符 1 / 常 量 1} {INTO / BY} {标识符 2 / 常 量 2} GIVING 标识符 3[,标识符 4]…

说明:

(1) 可以看到,当用 GIVING 部分时,第一个运算量(即标识符 1 或常量 1)可以除第二个运算量(标识符 2 或常数 2),如$\frac{B}{A}\Rightarrow C$,此时用 INTO,如 DIVIDE A INTO B GIVING C。也可以被第二个运算量除,如$\frac{A}{B}\Rightarrow C$,此时用 BY,如 DIVIDE A BY B GIVING C。结果总是放在 GIVING 后的标识符项中。二种用法不要弄错。也可以从英文句子的含义了解其用法。表 2.4 举例说明 DIVIDE 语句的用法和运算结果。

表 2.4

执行后结果 / 数据项 / 执行的语句	X (原值 40)	Y (原值 20)	Z (原值 10)
DIVIDE Z INTO X	4	20	10
DIVIDE Z INTO 100 GIVING Y	40	10	10
DIVIDE 2 INTO Z GIVING X	5	20	10
DIVIDE X BY Z GIVING Y	40	4	10
DIVIDE 60 BY Y GIVING X	3	20	10

(2) 允许 GIVING 后有几个标识符,如:

DIVIDE A BY B GIVING C, D, E

意为将 $\frac{A}{B}\Rightarrow C$, $\frac{A}{B}\Rightarrow D$, $\frac{A}{B}\Rightarrow E$。

(3) 如除不尽,则多余的位数截去,如$\frac{A}{B}\Rightarrow B$,若已定义 A 和 B 在内存中占两个字节,且为整数,如 A=6,B=4,则$\frac{6}{4}=1.5$,送到 B 时,由于 B 被指定为两个整数,不能容纳小数,所以 B 的值将为"01",而把小数点后面的部分截去。

下面是 DIVIDE 语句的一些例子:

DIVIDE TOTAL INTO AMOUNT GIVING AVERAGE.

```
DIVIDE AMOUNT BY TOTAL GIVING AVERAGE.
```

以上两个语句作用相同，都是计算：

$$AVERAGE=\frac{AMOUNT}{TOTAL}$$

```
DIVIDE 100 INTO PRICE GIVING RATE.
DIVIDE PRICE BY 100 GIVING RATE.
```

以上两个语句都是求：

$$RATE=\frac{PRICE}{100}$$

下面的用法是错误的：

DIVIDE A BY B	语句不完整，应有 GIVING 部分。
DIVIDE A BY 0 GIVING C	除数为零，溢出。
DIVIDE A INTO 3	不用 GIVING 部分时，INTO 后面不能跟常量。

2.3.5 四种算术语句的小结

上面介绍的四种算术(加、减、乘、除)语句，有以下特点：

(一) 一个语句只能进行一种单一的运算，不能在一个语句中实现两种不同的运算。如：A+B+C⇒D，可以用 ADD A,B,C GIVING D 实现。但若要计算 A−B+C⇒C 就要用两个语句：

```
ADD A TO C                  (A+C⇒C)
SUBTRACT B FROM C           (C−B⇒C)
```

来实现。

若要实现 $A^3+B^3\Rightarrow T$，则要以下几个语句：

```
MULTIPLY A BY A GIVING T    (A×A⇒T)
MULTIPLY A BY T             (A×T⇒T)
MULTIPLY B BY B GIVING P    (B×B⇒P)
MULTIPLY B BY P             (B×P⇒P)
ADD P TO T                  (P+T⇒T)
```

上面的 P 是中间数据项。

(二) 加法和减法语句可以进行两个以上数值量的计算，如 ADD A，B TO C 或 SUBTRACT A,B FROM C。但乘法、除法语句只能在两个量之间进行，即只能进行一次乘法或除法。如要实现 A×B×C，不能用：

```
    MULTIPLY A BY B BY C
```

或：MULTIPLY A，B BY C

以上两种写法都是错误的。而要用以下两语句：

```
MULTIPLY A BY B         (A×B⇒B)
MULTIPLY B BY C         (B×C⇒C)
```

即实现了 A×B×C⇒C。

(三) 四种算术语句，都有两种形式，即带 GIVING 部分和不带 GIVING 部分的。如果要求运算时不破坏所有参加运算的量时，可用格式 2(带 GIVING 部分)，把计算结果赋给另一个数据项。如不要求保留参加运算的量原来的值，则可用格式 1。

ADD 1 TO I	I 改变值。1+I⇒I
ADD 1 I GIVING J	I 值不改变，1+I⇒J

【例 2.2】

```
IDENTIFICATION DIVISION.          (标识部)
PROGRAM-ID. EXAM2-2.
ENVIRONMENT DIVISION.             (环境部)
DATA DIVISION.                    (数据部)
WORKING-STORAGE SECTION.          (工作单元节)
77 N PIC 99.
PROCEDURE DIVISION.               (过程部)
S.  ACCEPT N.                     | 1 | 2 |
    ADD 3 TO N.                   | 1 | 5 |
    DIVIDE 5 INTO N.              | 0 | 3 |
    SUBTRACT 2 FROM N.            | 0 | 1 |
    MULTIPLY 8 BY N.              | 0 | 8 |
    DISPLAY N.
    STOP RUN.
```

程序执行步骤：ACCEPT 是接收语句，等待从指定的设备（例如终端键盘）上接收数据给 N。在数据部中已说明 N 是两个字节的数据项，可放二位数字的整数。如果我们从键盘上输入数字 12，则在 N 中放了 12。第二个语句为 ADD 3 TO N，执行完后，N 值变为 15，在执行完第三个语句（除法语句）后，N 值变为 03。在执行完第四个语句（减法语句）后，N 值变为 01。在执行完第五个语句（乘法语句）后，N 的值为 08。第六个语句要求输出 N 的值，输出的结果是 08。然后程序停止运行。

【例 2.3】 一个工人每周工作六天，将每天实际工作时间相加，得一周总工时，乘以每小时的工资，得一周总工资，再乘上应上交的比例，得到应扣除部分，再从总工资中减去扣除部分，得到实发工资。

我们在这里只写出过程部中的有关语句。以后读者可以自己把它写成一个完整的 COBOL 程序。

```
    ⋮
ADD MON-HOURS, TUES-HOURS, WED-HOURS, THURS-HOURS, FRI-HOURS,
    SAT-HOURS GIVING TOTAL-HOURS.     (总工时)
    (将六天工时相加,"和"放在 TOTAL-HOURS 中)
MULTIPLY TOTAL-HOURS BY PAY-RATE
          (总工时)      (每小时工资)
    GIVING GROSS-PAY.
          (总工资)
MULTIPLY GROSS-PAY BY RATE GIVING WITHHOLDING.
          (总工资)      (比率)          (扣除)
SUBTRACT WITHHOLDING FROM GROSS-PAY
          (扣除)                (总工资)
    GIVING NET-PAY.
      (实发工资)
    ⋮
```

⋮

读者应能看懂这个程序片断。

2.3.6 计算语句(COMPUTE 语句)

用以上四种语句进行比较复杂的计算是不方便的，如 $A^3+B^3\Rightarrow T$，就要五个语句。COBOL 提供了一种计算语句(COMPUTE)，可以进行比较复杂的四则运算。如想计算 T=(A+B)×C÷D，只要写一个 COMPUTE 语句：

```
COMPUTE  T  =  (A  +  B)  *  C  /  D
```

它还可以进行乘方的运算。

(一) 算术表达式

1. 什么是算术表达式

算术表达式是由：(1)算术初等量(数值常量、数值型数据项)；(2)算术运算符(+，-，*，/，* *)；(3)括号，所组成的有意义的式子。如上面(A+B)*C/D 就是一个符合 COBOL 规定的算术表达式。表达式中可以包括括号，也可以没有括号，根据需要决定。

注意，表达式中不应包括等号，不要把 T=(A+B)*C/D 说成是表达式。等号右边的才是表达式。

2. 运算次序

在表达式中包括各种运算符和括号，它们按以下优先次序进行计算：

(1) 括号( )	高	运算优先级为上高下低
(2) 单目运算符(正负号)		
(3) 乘方 (* *)		
(4) 乘(*)，除(/)		
(5) 加(+)，减(-)	低	

同等级的运算符按自左而右次序。如

```
-Z * * 2 / 2 + 1
 ①
   ②
     ③
       ④
```

运算次序为：①-Z 为单边运算符运算，先对 Z 取负值。②乘方，进行$(-Z)^2$计算。③除以 2。④加 1。

表 2.5 给出了一些数学表达式和 COBOL 中表达式的对照关系：

表 2.5

数学表达式	COBOL 中表达式
$W-\frac{A+B}{C+D}$	W - (A + B) / (C + D)
X^2+X-3	X * * 2 + X - 3
$\frac{B}{A}+C(X+1)^2$	B / A + C * (X + 1) * * 2
$-A^2-A$	- (A * * 2) - A
$\frac{1}{\frac{1}{R_1}+\frac{1}{R_2}}$	1 / (1 / R1 + 1 / R2)

以下的对应关系是不正确的：

数学表达式	COBOL 表达式
$\frac{A+B}{C+D}$	A+B/C+D
$\frac{A}{-B^2}$	A/−B * * 2

在写表达式时注意：

(1) 所有运算符两侧均应留一空格，如：A=B+C，应写成 A = B+C

(2) 括号的外侧应留空格，内侧可不要留空格。

(二) 计算语句的一般格式

```
COMPUTE 标识符 1 [,标识符 2]…=算术表达式
```

```
如：  COMPUTE   A=B * * 2 + C * X
      COMPUTE   T=A
      COMPUTE   P=3
      COMPUTE   A,B=3 * 5   它的作用是 3×5⇒A，3×5⇒B。将表达式 3 * 5 的值分别赋
                            给 A 和 B。
```

计算语句虽然在计算表达式时书写方便，但它的运算速度较慢。如 COMPUTE C=A+B 比 ADD A,B GIVING C 慢些。由于 COBOL 主要不用于科学计算，很少遇到复杂的表达式，因此并不常用到 COMPUTE 语句。

(三) 举例

我们尽可能地逐步介绍一些简单的程序，以帮助读者尽早地熟悉 COBOL 程序的编写方法。由于尚未介绍 COBOL 的前三部分(标识部、环境部、数据部)，因此，对这几部分中的一些细节，可以先“不求甚解”，有一个概念即可。在以后介绍到这几部分时，就不会感到陌生了。这样可以起到一举两得的作用。

【例 2.4】

```
IDENTIFICATION DIVISION.          (标识部)
PROGRAM-ID. EXAM2-4.
ENVIRONMENT DIVISION.             (环境部)
DATA DIVISION.                    (数据部)
WORKING-STORAGE SECTION.          (工作单元节)
77  X  PIC  9999.
77  A  PIC    99.
77  B  PIC    99.
77  C  PIC    99.
PROCEDURE DIVISION.               (过程部)
S. ACCEPT A.
   ACCEPT B.
   ACCEPT C.
   COMPUTE  X=(A+B)/C.
   DISPLAY  X.
   STOP RUN.
```

说明：从系统指定的设备(假设指定为从终端键盘)上接收 A,B,C 三个数据(A,B,C 已在数据部中定义为两位整数)。由键盘输入 A 的值(即 15)，然后按“回车”键，再在第二行上

输入 B 值 05。(注意 B=5,但不能输入␣5↙,因空格不是数字),按"回车"键,再输入 C04 的值,按"回车"键。即:

15↙
05↙
04↙

计算语句计算出 X 的值为(15+5)/4=5,而 X 在数据部中已被定义为 4 位整数,因此它在内存中四个字节放了 0005,执行 DISPLAY 语句时显示出"0005"四个字符。

【例 2.5】 不用 DISPLAY 语句显示结果,而用 WRITE 语句在打印机输出。

```
IDENTIFICATION  DIVISION.  (标识部)
PROGRAM ID.  EXAM2 5.
ENVIRONMENT  DIVISION.          (环境部)
INPUT-OUTPUT  SECTION.          (输入输出节)
FILE-CONTROL.                   (文件控制段)
  SELECT  OUTPUT-FILE  ASSIGN  TO  打印机名.
    (使文件 OUTPUT-FILE 与打印机联系)
DATA  DIVISION.
FILE  SECTION.
FD  OUTPUT-FILE  LABEL  RECORD  IS  OMITTED.
01  OUTPUT-RECORD.        (OUTPUT-RECORD 属于 OUTPUT FILE 文件)
  02  FILLER     PIC  X.          (走纸控制位)
  02  OUT-REC  PIC  X(120).      (假设所用打字机每行容纳 120 个字符)
WORKING STORAGE  SECTION.    (工作单元节)
77  T  PIC  9999.
01  A.
  02  A1  PIC  9(4).
  02  A2  PIC  9(4).
  02  A3  PIC  9(4).
PROCEDURE  DIVISION.            (过程部)
OPEN  OUTPUT  OUTPUT FILE.  (打开所用文件)
ACCEPT  A.
COMPUTE  T  =  (A1  +  A2)  /  A3.
MOVE  A  TO  OUT-REC.
WRITE  OUTPUT-RECORD  AFTER  1.
MOVE  T  TO  OUT-REC.
WRITE  OUTPUT-RECORD  AFTER  1.
CLOSE  OUTPUT-FILE.
STOP  RUN.
```

在本例中,列出了 COBOL 程序的四大部分,以使读者有一个整体的概念。在本章中我们只着重分析过程部,对其它部分只作简单的介绍。

在数据部中,输出记录 OUTPUT-RECORD 中包含两个初等项,第一个初等项 FILLER 的作用只是占用一个字符,以便作为纵向走纸控制之用。这个数据项并不准备输出,因此它的名字并不重要,一般采用保留字 FILLER(填充项)。真正要输出的内容应放在 02 层的 OUT-REC 中。

工作单元节用来定义不属于文件的数据。A 为组合项,下属三个初等项 A1、A2、A3。每个数据项为 4 位数字。

如果从键盘上输入给数据项 A 的数据为 010002000003,则 A 中的各数据项中存放的数据如下:

A		
A1	A2	A3
0100	0200	0003

由 COMPUTE 语句计算出 T＝(100＋200)/3＝100，第一次执行 WRITE 语句时输出 A 的值，即：

010002000003

第二次执行 WRITE 语句时输出 T 的值：

0100

由于 OUT-REC 是记录 OUTPUT-RECORD 的一部分(实际上是除了第一个字符 纵向走纸控制字符 以外的全部内容)，而 OUTPUT-RECORD 又是 OUTPUT-FILE 文件的记录，OUTPUT-FILE 又与打印机相联系，因此执行 WRITE OUTPUT-RECORD 就在打印机上输出 OUT-REC 的全部内容。

2.4 传送语句(MOVE 语句)

2.4.1 传送语句的作用

MOVE 语句用来实现数据的传送，将一个数据从一个内存域送到另一个内存域中。注意，是数据在内存中的传送，而不是内存和外部设备之间的传送。MOVE 语句是 COBOL 中用得最广泛的语句之一。相当于其它高级语言中的赋值语言。假设 A＝12，B＝123，执行 MOVE A TO B 的前后，在内存中 A 和 B 的内容如图 2.13 所示。

执行前：

A		
0	1	2

B		
1	2	3

执行后：

A		
0	1	2

B		
0	1	2

图 2.13

即将 A 的值传送给 B，B 值改变了，而 A 值不变。注意，B 原来的值被“冲掉”，而代以新值，将 A 的值送入 B 中，取代了 B 的原值。例如：

```
MOVE AMOUNT TO TOTAL.
MOVE 1285 TO A.
MOVE ZERO TO COUNT.
MOVE SPACE TO BLANKFIELD.
MOVE 'BOOK' TO T.
MOVE '1981/8/30' TO W, X, Y.
```

即可以将常量(包括数值常量、非数值常量、表意常量)或一数据项的内容传送给另一数据项。

2.4.2 传送规则

MOVE A TO B

其中 A 称为发送项，B 称为接收项。

(1) 如果接收项和发送项在数据部中描述的类型和长度相同，则按字节一一对应地传送。如上面介绍的例子。

(2) 如果接收项与发送项长度不相同，而二者都是数值数据项，则按"小数点对齐"原则处理。如果是整数，则认为小数点在最后一位数字之后。如接收项长度大于发送项，则多余位补零。如接收项长度小于发送项，则产生截断。见图 2.14。

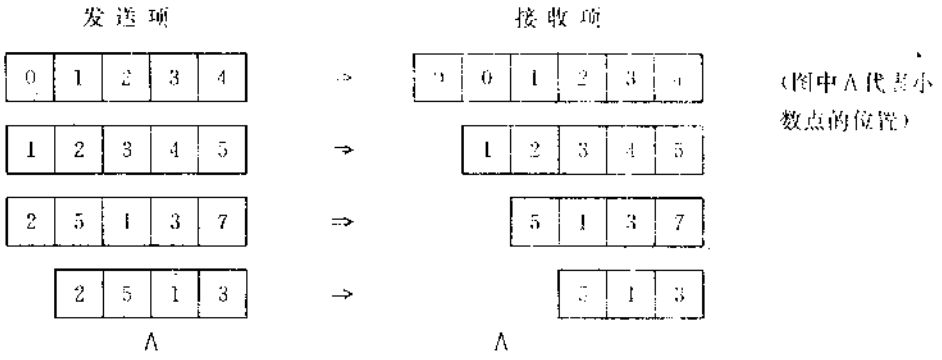


图 2.14

(3) 对字母或字符数据(非数值型数据)的传送，按'左对齐'原则处理。如接收项长度大于发送项的长度，则多余位置填空格。如接收项的长度小于发送项的长度，则从右端截断，将多余的字符被舍去。见图 2.15。

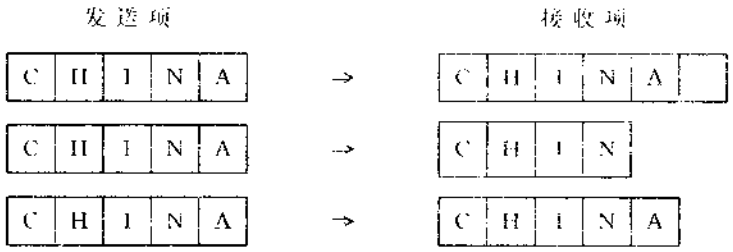


图 2.15

(4) MOVE 语句可以将一初等项内容传送给另一初等数据项，也可以将一组合项内容传送给一初等数据项，也可以将一初等项内容传送给一组合项。设 A 和 B 都是组合项，如图 2.16 所示。

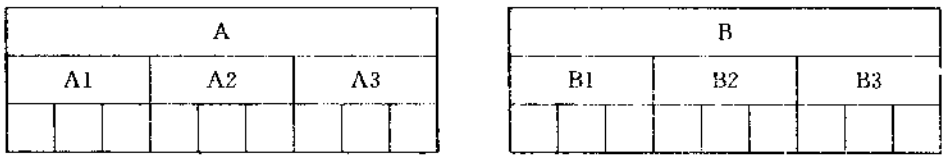


图 2.16

则：MOVE A TO B
MOVE A1 TO B1
MOVE A2 TO B2
MOVE A3 TO B3

都是合法的。MOVE A TO B 相当于一次将 A 所占的内存域的全部内容传送给 B。

若 A 是组合项，T 是初等项。图 2.17 的传送是合法的：

MOVE A TO T

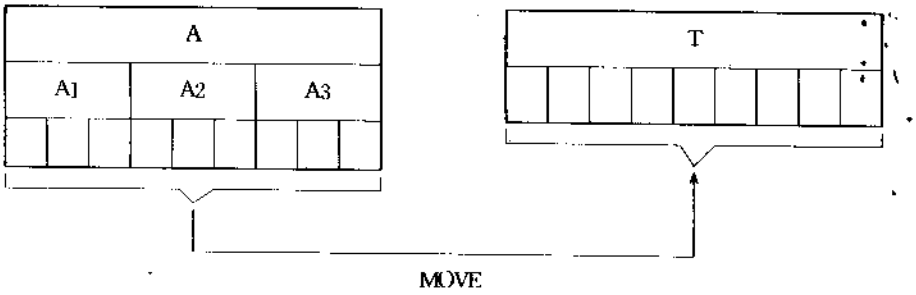


图 2.17

2.4.3 MOVE 语句的一般格式

$$\underline{\text{MOVE}}\left\{\begin{matrix}\text{标识符 1}\\ \text{常 量 1}\end{matrix}\right\}\underline{\text{TO}}\ \text{标识符 2}\ [,\text{标识符 3}]\cdots$$

表 2.6 举例说明了 MOVE 语句的使用。

表 2.6

传送后结果 / 数据项 / 执行的语句	X (原值 10)	Y (原值 15)	Z (原值 25)
MOVE X TO Y	10	10	25
MOVE 15 TO X	15	15	25
MOVE Z TO Y	10	25	25
MOVE X TO Y,Z	10	10	10
MOVE ZERO TO X, Y, Z	00	00	00

在第五章中还将介绍各类型数据之间的传送规则。

§2.5 转移语句(GOTO 语句)

有时需要使程序改变正常执行的顺序，这时可用 GO TO 语句。这是一个无条件转移语句。程序执行到此语句时，无条件地转到指定的节或段去。如：

A1. ACCEPT B.
ACCEPT C.
ADD B TO C.

```
A2. DISPLAY  C.
    GO  TO  A1.
```

A1 和 A2 是过程部中的段名。这段程序的作用是：接收 B 和 C 的值，使 B+C⇒C，然后显示 C 的值。回到 A1 段，再接收两个新的值给 B,C，然后再计算和显示一次，周而复始。计算若干组结果。这个程序是不会自动停止的。一直循环下去。

只能转移到段(或节)的开头，不能转移到段的当中某一语句(如上面的 ACCEPT C 语句)。节名和段名称为过程名。它代表一段过程。

一般格式：

格式 1

```
GO  TO  过程名
```

格式 2

```
GO  TO  过程名 1 [,过程名 2]…过程名 n  DEPENDING  ON  标识符
```

格式 2 的 GO TO 语句是根据标识符的值来决定应转到什么地方的，如：

```
GO TO A, B, C, D, E DEPENDING ON I
```

当 I=1 时，转到 A 段，I=2 时，转到 B 段，I=3 时，转到 C 段，依此类推。I=n 时，转到 GO TO 后面的第 n 个过程名所指定出的节(段)去。显然，如果 GO TO 后面只提供五个过程的名字，则 I 的值应 1≤I≤5，否则执行此语句的下一个语句。

【例 2.6】 银行有几种不同的利率(6%，5%，4%，2.5%)，对不同的存户根据其特点确定一个利率并计算利息。

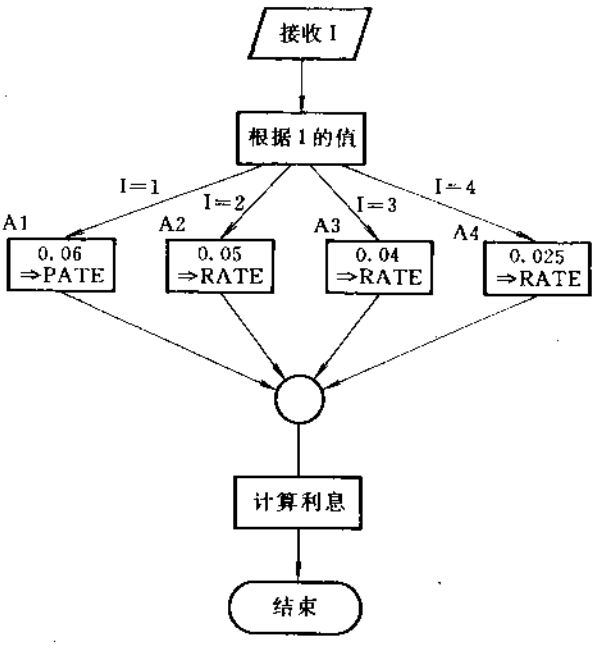


图 2.18

```
    ⋮
    ACCEPT I.
    GO TO A1, A2, A3, A4 DEPENDING I.
A1. MOVE 0.06 TO RATE.
    GO TO B.
A2. MOVE 0.05 TO RATE.
    GO TO B.
A3. MOVE 0.04 TO RATE.
    GO TO B.
A4. MOVE 0.025 TO RATE.
B. COMPUTE INTEREST=PRINCIPAL * RATE.
                (利息)    (本金)      (利率)
   DISPLAY 'INTEREST=', INTEREST.
   STOP RUN.
```

流程见图2.18。如输入1给I，则根据I的值由GO TO语句使流程转到A1段，使利率定为6%，然后转到B段，计算利率为6%时的利息。如输入给I的值为3，则转去A3，利率为4%，计算利息。

【例2.7】 运输公司为招揽业务，对每次运输货物在50吨以下的，按每吨10元收费。50吨以上(含50吨，下同)100吨以下，每吨收费9元。100吨以上150吨以下的，每吨收费8元。150吨以上200吨以下，每吨收费7元。200吨以上250吨以下，每吨6元。如已知吨数不超过250吨，求运费。

程序如下，相应的流程图见图2.19。

```
IDENTIFICATION DIVISION.                (标识部)
PROGRAM-ID. EXAM2 7.
ENVIRONMENT DIVISION.                   (环境部)
DATA DIVISION.                          (数据部)
WORKING-STORAGE SECTION.                (工作单元节)
77  M   PIC          9999.
77  I   PIC          9.
77  AMOUNT PIC   99999.
PROCEDURE DIVISION.                     (过程部)
    ACCEPT   M.
    DIVIDE 50 INTO M GIVING I.
    ADD 1 TO I.
    GO TO A. B. C. D. E DEPENDING I.
A. COMPUTE AMOUNT = 10 * M.
   GO TO F.
B. COMPUTE   AMOUNT = 9   *   M.
   GO TO F.
C. COMPUTE   AMOUNT = 8   *   M.
   GO TO F.
D. COMPUTE   AMOUNT = 7   *   M
   GO TO F.
E. COMPUTE   AMOUNT = 6   *   M
F. DISPLAY M, AMOUNT.
   STOP RUN.
```

如输入M(吨数)为140吨，则M除以50后送到I的值为2(商的整数部分为2，多余数舍去)。然后I+1⇒I，即I等于3，在GO TO语句中决定程序应转去执行C段，每吨乘8元，然后显示出吨数和金额。当M<50时，M/50⇒I后，I的值为0，经过ADD 1 TO I后，I=1，转A

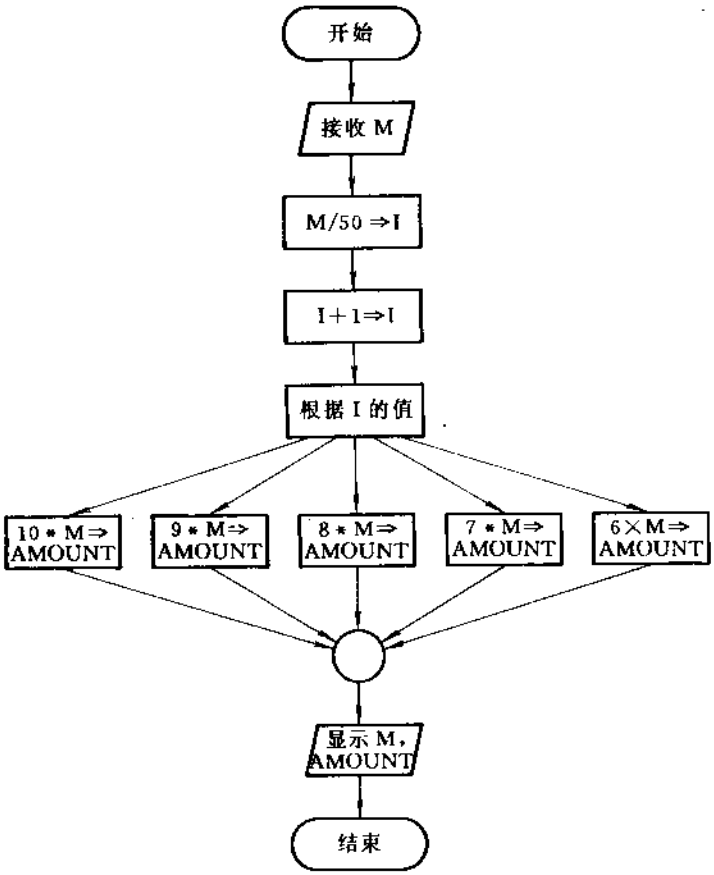


图 2.19

段，每吨乘10元。

ADD 1 TO I 语句的作用在于使 I 的值不小于1，否则 GO TO 语句无效。

这个题目可以这样思考：以50吨为一档，当重量为50吨的倍数时单位运价就发生变化。可由表2.7表示。

表 2.7

重 量 (M)	有几个50吨	I	单位运价(p)
0<M<50	0	1	10
50≤M<100	1	2	9
100≤M<150	2	3	8
150≤M<200	3	4	7
200≤M<250	4	5	6

上表中 I 的值等于 M/50(取整)加1。显然，应当根据 I 的值转到不同的段中，在这些段中按不同的单位运价计算运费。

如果把题目要求改变一下：小于100吨的每吨收费9元。其余不变，请读者考虑程序如何

修改?

此时可以将表2.7改为表2.8。

表 2.8

重 量 (M)	有几个50吨	I	单位运价
0<M<50	0	1	9
50≤M<100	1	2	9
100≤M<150	2	3	8
150≤M<200	3	4	7
200≤M<250	4	5	6

只需将上面程序中的 GO TO 语句改为:

GO TO B, B, C, D, E DEPENDING I.

并取消程序中的 A 段即可。也就是当 I=1和2时都转去 B 段处理。

这种 GO TO 语句的作用好比一个多路开关,当开关旋钮在某一位置时,接通一个电路,见图2.20。因此可把它称为开关语句。

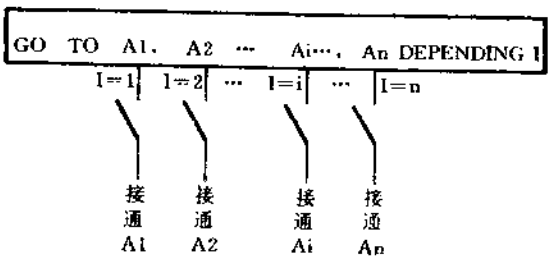


图 2.20

请注意程序中每一个段的最后都有一个"GO TO F"语句,以便汇合在一个出口处,然后结束。请对照流程图2.19研究;如果没有这些 GO TO 语句行不行?

【例2.8】 将上题改由打印机输出结果。

```
IDENTIFICATION  DIVISION.          (标识部)
PROGRAM-ID.     EAXM 2-8.
ENVIRONMENT     DIVISION.          (环境部)
INPUT-OUTPUT    SECTION.
FILE-CONTROL.
    SELECT  PRINT-FILE  ASSIGN  TO  PRINTER.
DATA  DIVISION.                    (数据部)
FILE  SECTION.
FD  PRINT-FILE  LABEL  RECORD  IS  OMITTED.
01  PRINT-RECORD.
  02  FILLER  PIC  X.
  02  PRINT-REC  PIC  X(120).
WORKING-STORAGE  SECTION.
77  M  PIC  9(4).
77  I  PIC  9.
```

```
77  PRICE  PIC  99.
77  AMOUNT  PIC  9(5).
PROCEDURE  DIVISION.                    (过程部)
  ACCEPT  M.
  DIVIDE  50  INTO  M  GIVING  I.
  ADD  1  TO  I.
  GO TO  A,B,C,D,E  DEPENDING  I.
A.  MOVE  10  TO  PRICE.
    GO  TO  F.
B.  MOVE  9  TO  PRICE.
    GO TO  F.
C.  MOVE  8  TO  PRICE.
    GO TO  F.
D.  MOVE  7  TO  PRICE.
    GOTO  F.
E.  MOVE  6  TO  PRICE
F.  COMPUTE  AMOUNT=PRICE  *  M.
    MOVE  AMOUNT  TO  PRINT-REC.
    WRITE  PRINT-PECORD  AFTER  2.
    CLOSE  PRINT-FILE.
    STOP  RUN.
```

请读者自己消化理解此程序。本程序中以数据项 PRICE 代表单位运价。若在运行时从键盘输入：

```
0180↙      (注意输入4位数字)
01260      (运费为1260元)
```

§2.6 条件语句(IF 语句)

程序执行时，一般是按语句的先后顺序执行的，但实际上在数据处理中，常常需要根据给出的某些条件是否满足来决定应执行那一部分语句。例如，银行希望把款额大于1000元的存户名字和存款数打印出来，在商业来往帐目中希望把负债的(结余金额为负的)单位打印出来，人事部门希望把女职工名单或工程师名单打印出来，等等。这就需要由程序编制者事先给出一个条件(如存款大于1000元)，然后在程序执行时，判断这些条件是否满足？如果满足，则执行某一些语句(如打印出某些内容)，如不满足，则执行另一些语句。如：

```
IF AMOUNT IS GREATER THAN 1000
  DISPLAY NAME, AMOUNT.
IF SEX IS EQUAL TO 'F' DISPLAY NAME, SEX.
IF X<0 MOVE X TO T.
```

第一个语句的含义是：如果数据项 AMOUNT 的值大于1000，则显示出数据项 NAME(存户名字)和 AMOUNT(金额)的值。第二个语句的含义是：如果数据项 SEX 的值等于"F"，则显示出 NAME 和 SEX 的值。第三个语句的含义是：如果 X 的值小于0，则将 X 的值传送给另一数据项 T。

2.6.1 关系运算符

为了给出条件，一般需用到关系运算符，如 X=0，A>B，等。COBOL 规定的关系运算符如表2.9。

表 2.9

COBOL 关系运算符	意　义	相当数学上的符号
IS GREATER THAN ＞	大　于	＞
IS LESS THAN ＜	小　于	＜
IS EQUAL TO ＝	等　于	＝
NOT GREATER THAN NOT ＞	不大于	≯
NOT LESS THAN NOT ＜	不小于	≮
NOT EQUAL TO NOT ＝	不等于	≠

“关系运算符”，它的作用是对两个数值型或字符型数据的大小进行比较。

例如：IF A NOT GREATER THAN 0 DISPLAY A.

和　IF A NOT＞0 DISPLAY A.

都表示：如果满足“A 不大于0”条件(即 A≤0)时，将 A 的值显示出来。

2.6.2　关系运算规则

两个量进行关系运算，按以下规则：

(1) 数值量之间的比较。按他们的代数值进行比较(即按有效数和代数符号)。如：

18.5和18.50，+18.500，0018.5　都相等。而　17.5＞－17.5，＋100＞－200。

(2) 字母型数据的比较。如果有两个非数值常量“A”和“B”相比，谁大？在计算机中，字符都用一组数字代码来代表，不同的计算机所采取的代码不同，如有的计算机用 ASCII 代码，A 的代码是八进制的101，B 的代码是102，则认为“A”＜“B”，因为按他们的代码大小进行比较。有的计算机用 EBCDIC(Extended Binary Coded Decimal Interchange Code)码，A 的代码是二进制的11000001，B 是11000010，因此，B 也是大于 A。不论用那一种代码，有一共同规律，按 A 到 Z 次序，后面的字母值比前面的大。即 Z＞Y＞X……＞A。

如果是由几个字母组成的非数值常量，如：“BOY”和“BOT”谁大？将它们自左而右地逐个字母进行比较，直到出现第一个不同的字母为止。例如这两个字前两个字母都是“B”和“O”，相同。第三个字母不同，而按前面说的规则，“Y”大于“T”。因此认为“BOY”＞“BOT”。如果是“THAT”和“THEN”比较，前二个字母相同，第三个字母比较“A”＜“E”，而第四个字母比较“T”＞“N”。则以出现的第一个不同的字母比较结果为准，而不管以后的字母比较结果如何。因此“THAT”＜“THEN”，是根据“E”＞“A”得出的结论。

如果两个字母型的数据长度不同，如“BOOK”和“BOOKS”比较。将短的数据右边补空格，使二者等长，将“BOOK”补空格成“BOOK ␣”，再与“BOOKS”比，“␣”的代码比“A”还小，即它比任何字母都小，所以“BOOK”＜“BOOKS”。

以上的比较规则可以用一句话来概括：按英文字典顺序，在英文字典中位置在后的字比

位置在前的字大，如“CAT”和“DOG”，英文字典中“DOG”在后，因此“CAT”<“DOG”。又如“THE”和“THESE”，在字典中“THESE”在后，因此“THE”<“THESE”。

(3) 字符型数据的比较。参加比较的数据不是数值型，又不纯是字母，而是由任意的字符组成，如“A42”和“9B$”二者谁大？同样自左而右逐个字符相比，按字符代码大小进行比较。不同的计算机采用的字符代码不同，因而字符“大小”的次序也不相同。例如按 ASCII 代码比，则“A”>“9”，而按 EBCDIC 代码比，则“9”>“A”。标准 COBOL 对用什么代码不作统一规定，由各计算机系统确定。如用到这方面的比较时，应查本计算机系统说明书。如 VAX，IBM-PC 系列和长城0520系列机用的 COBOL，采用 ASCII 代码，而 M 系列中型机（如 M150）采用 EDCDIC 代码。

2.6.3 IF 语句的两种形式

利用 IF 语句进行程序设计是程序设计的基本技巧之一。在许多程序中都要求根据某个给定的条件进行判断选择处理，即从两个或多个处理方案中选定一种。用 IF 语句来实现这种要求是很方便的。

IF 语句有两种形式

(一) IF 条件 语句组

例如，某公司对顾客购买商品1000件以上的，给3%的优惠，可写成以下句子。

```
IF  QUANTITY IS NOT LESS THAN 1000
    MULTIPLY 0.97 BY PRICE.
MULTIPLY QUANTITY BY PRICE GIVING TOTAL.
```

流程图见图2.21。

这种 IF 语句的形式是在两个分支路径中的一个分支包含一个或多个可执行语句（即语句组），而在另一个分支中不执行任何操作。见图2.22。

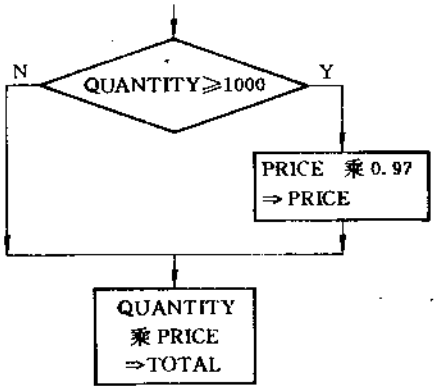


图 2.21

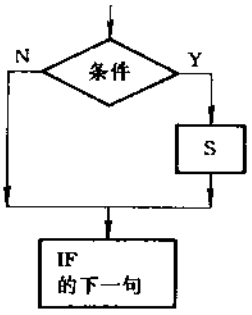


图 2.22

(二) IF 条件 语句组1 ELSE 语句组2

有时希望当条件满足时执行某一组语句，不满足则执行另一组语句，见图2.23。这时可用 IF-ELSE 形式的条件语句。

如果把上面例子改成：当买商品1000个以上者减价3%，在1000个以下的减价1%。流程

图见图2.24。可以用下面的 IF 语句实现。

```
IF  QUANTITY  <  1000
    MULTITY  0.99  BY  PRICE
ELSE
    MULTITY  0.97  BY  PRICE.
MULTIPLY  PRICE  BY  QUANTITY  GIUING  TOTAL.
```

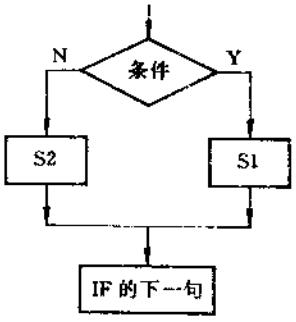


图 2.23

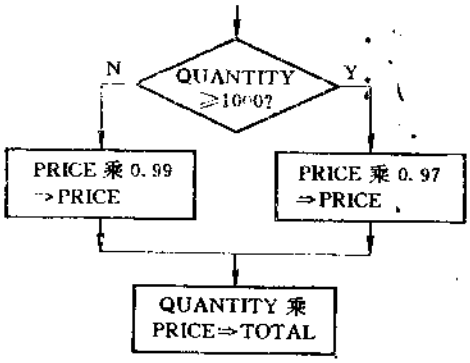


图 2.24

它的作用是：先判断给定的条件"QUANTITY＜1000"是否满足，若满足则执行将 PRICE 乘0.99的乘法语句，否则执行 PRICE 乘0.97。不论是乘0.97或0.99，都接着执行 IF 语句的下一个语句。

在这里，我们强调一下句子(Sentence)和语句(statement)的区别。句子可以包括若干个语句，句子以句点和空格结束。在以前遇到过的简单程序中，句子和语句好像没有多大区别，例如：

```
(1) MOVE A TO B      (2) MOVE A TO B.
    MOVE C TO D          MOVE C TO D.
    MOVE E TO F.         MOVE E TO F.
```

左边第(1)组中前两个语句后无句点，只有第三个语句后有一个句点，因此这三个语句属于一个句子。而右边第(2)组中，每一个语句后都有一个句点，它们是三个句子。在执行时，这两种情况没有什么区别，都是依次执行这三个语句。

但是用到 IF 语句时，句点加在不同的地方作用往往是不相同的。一个句子是以句点和空格结束的。一个 IF 语句的范围是遇到句点和空格为止。当 IF 语句中指定的条件不满足时就应当执行该句点和空格后面的下一个语句。例如：

```
IF  A>0  DISPLAY  A
ADD  A  TO  TOTAL.
DISPLAY  TOTAL.
STOP  RUN.
```

在第二行最后有一个句点和一个以上的空格，表示 IF 语句到此结束。由于出现句点与空格，也表示一个句子到此结束。因此，一个 IF 语句同时是一个 IF 句子。

当 A≤0时，不执行 DISPLAY A 和 ADD A TO TOTAL 这两个语句(它们是属于 IF 语句的"内嵌语句")，而执行第三行的 DISPLAY TOTAL。当 A＞0时，显示 A 的值，并将 A 加到 TOTAL 中，见图2.25。

如果在第二行后没有句点，则认为第三行的 DISPLAY TOTAL 也是 IF 语句中的内嵌语句，如图2.26所示。当 A≤0时，就不执行 DISPLAY TOTAL 而直接执行 IF 句子后面的语句(即 STOP RUN)了。意思就完全不同了。

如果在第一行后加一句点，则认为整个 IF 句子到此结束，见图2.27。

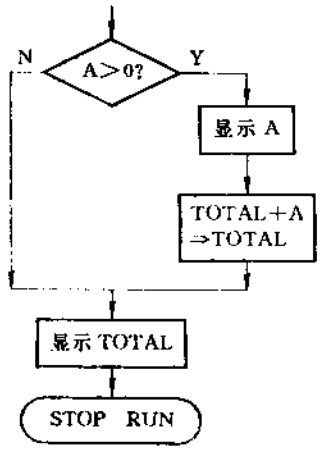


图 2.25

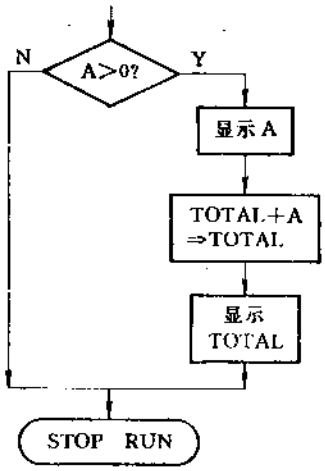


图 2.26

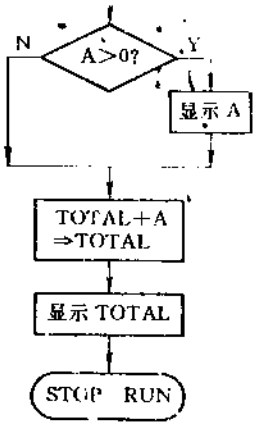


图 2.27

通过以上三种情况的对比，可以看到在使用 IF 语句时正确使用句点的重要性。不少初学者常常在这方面出错。

如果用 IF-ELSE 形式：

```
IF  条件  语句组1
ELSE  语句组2.
```

语句组1和语句组2都可以包含一个或多个语句。如何检查语句组1或语句组2到什么地方为止呢?这些语句组包含一系列语句直到遇到 ELSE 或遇到句点和空格为止。

2.6.4 IF 语句的一般格式

```
IF 条件 {语句组1 | NEXT SENTENCE} [ELSE {语句组2 | NEXT SENTENCE}]
```

除了已介绍过的用法外，还有一种 NEXT SENTENCE 的用法。即条件满足(或不满足时)执行"IF 下面的句子"。如：

```
IF  B * * 2  -  4  *  A  *  C  NOT  <  0
    NEXT SENTENCE
ELSE
    DISPLAY 'B  * *  2  -  4  *  A  *  C  <  0'.
STOP  RUN.
```

下面两个 IF 语句等价：

(1) IF A>B

```
    MOVE A TO B
  ELSE
    NEXT SENTENCE
(2) IF A>B
    MOVE A TO B.
```

2.6.5 IF 语句应用举例

【例2.9】 我们把本章例2.3变化一下：某集体单位统计职工每周的工作时间，如规定应出勤工时为48小时，超过48小时的部分，按加班处理。加班的工资为平常工资的1.5倍，计算出总工资，再扣除应上交的金额，可得到实发工资。

只写过程部中有关语句：

```
    ⋮
ADD MON-HOURS, TUES-HOURS, WED-HOURS,
    THUS-HOURS, FRI HOURS, SAT-HOURS
    GIVING TOTAL HOURS.  (总工时)
MULTIPLY TOTAL HOURS(总工时) BY PAY-RATE(每小时工资)
  GIVING GROSS-PAY.(总工资)
IF TOTAL-HOURS > 48
  MULTIPLY PAY-RATE BY 0.5
    GIVING OVERTIME-RATE(每小时外加的加班工资)
  SUBTRACT 48 FROM TOTAL-HOURS(总工时)
    GIVING OVERTIME-ROURS(加班工时)
  MULTIPLY OVERTIME HOURS(加班工时)
    BY OVHRTIME-RATE(外加的加班工资/时)
    GIVING OVERTIME-PAY(加班工资)
    ADD OVERTIME-PAY TO GROSS-PAY.
MULTIPLY GROSS-PAY BY RATE(上交比率)
      GIVING WITHHOLDING.(应扣除款)
SUBTRACT WITHHOLDING FROM GROSS-PAY(总工资)
      GIVING NET PAY.(实发工资)
```

读者可以自己看懂这段程序。其中的各数据名取的是有含义的英文字，以使程序意思一目了然。读者可以自己将程序写完全。

§2.7 停止语句(STOP)语句

当实现了程序预期的要求后，应使程序停止执行。

一般格式为：

```
STOP { RUN }
     { 常 量 }
```

说明：(1) 当用 STOP RUN 时，执行此语句将程序停止运行，停止后不能再接着往下运行。如有需要，可再重新运行一次。

(2) 用 STOP 常量，表示程序暂时挂起不往下执行，显示出此常量(可以是数值常量或非数值常量或表意常量)。如有以下语句：

```
STOP 123.
```

在执行到此语句时，程序暂停，并显示出123字样。

此功能常用于调试程序。可以分段调试程序，中间设一些“中断点”。程序运行到某一处暂停，以便程序员检查运行情况。暂停后还可以从键盘打入适当的命令（在不同的系统中规定了不同的命令），使程序继续运行。

§2.8 结构化程序设计要点

2.8.1 程序设计的概念

在本章中介绍了一些基本的过程部语句，读者已经可以编写和运行一些简单的COBOL程序了。我们知道，设计一个程序最重要的工作有两个方面，即设计数据结构和设计操作步骤。著名计算机科学家 wirth 提出过一个著名的公式：数据结构＋算法＝程序。数据结构表示数据的组织形式。操作步骤则决定计算机如何进行操作。好像一个菜谱包括原料和操作步骤这两个部分，做菜肴就是对给定的原料进行加工，二者缺一不可。程序加工的对象是数据，首先要想好数据之间的相互联系关系，例如是简单的数据项还是数组、是组合项还是初等项，互相间的依存关系等等，这些将主要在数据部中定义。

在过程部中主要解决操作步骤问题。在写过程部语句之前，应先设计好操作过程（执行什么操作以及操作的次序等）。这项工作一般称为算法（algorithm）设计。在程序设计中，常用“算法”这一术语。所谓算法是指“解题方法的精确描述”。算法可以用流程图来表示，也可以用自然语言表示，但由于自然语言往往显得冗长而且有“歧义性”，因此一般不采用自然语言来表示算法。

2.8.2 结构化程序设计的概念

结构化程序设计是指编制程序的一种方法。我们知道，程序设计最起码的要求是设计一个能在计算机上运行的，能得到正确结果的程序。但是仅仅做到这一点是远远不够的。在计算机发展的初期程序设计的方式是手工业式的。程序的编写没有什么规范可以遵循，全靠程序设计人员的个人习惯和技巧。因此，基于程序员个人的技术和素质而产生的程序设计风格是因人而异的。在程序中充分体现了程序员的个性，素质及聪明才智和技技巧的娴熟。这是程序员喜欢津津乐道并为人们赞赏的一面。然而另一方面却包含了一种巨大的潜在弊端。由于在传统程序中程序员无约束地使用 GO TO 语句，使得程序语句的书写顺序和程序语句的执行顺序不一致。从程序执行角度上观察，按书写顺序的那些程序语句象面条一样互相纠缠在一起，这种情况如图2.28所示。

这种“一碗面条式样”的程序不仅使人们难于读懂，就是程序员本人在相隔一段时间后，重新阅读自己编写的程序也会感到十分困难。随着计算机技术的发展，60年代末到70年代初，软件逐渐发展成一种行业，从事软件开发的人员越来越多，程序已不再是个人创造的“工艺品”，而逐渐变成了一种大生产下的“软件产品”，往往需要许多人共同来完成一个软件产品，这就要寻找一种标准化的、规范化的产品式的程序设计方法，使程序设计者按照这种规范行事，写出具有良好结构的程序。这种程序易于编写，易于阅读，易于调试，易于修改维护。在这种历史条件下，传统的程序设计方法受到了挑战。程序不应是随心所欲的产物，而应是象工业生产那样，采取工程化的方法，不受个人性格或爱好的影响，使不同的人编写程序都遵循同一个方法与规范。现在衡量程序质量的标准已不再是“效率第一”了，而是“清晰第一、

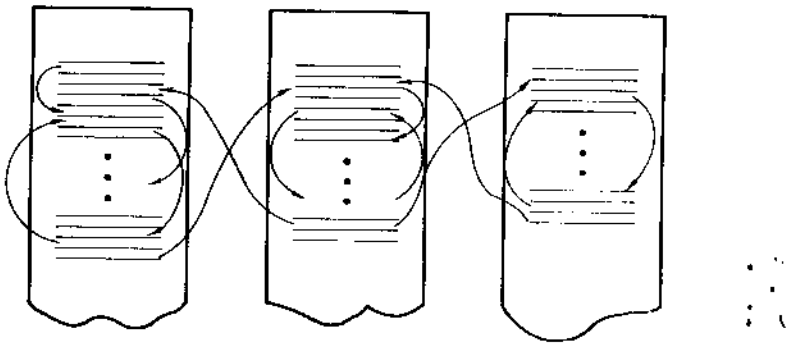


图 2.28

效率第一"。因为在硬件成本大大降低、内存量已大大增加、运算速度大大提高的今天，节约一点点内存量和时间所换得的好处是微不足道的，而从事软件的人才的时间才是最宝贵的，只有程序结构清晰，才能做到程序易于阅读、易于调试、易于修改、易于移植、易于交流。这种程序也易于编写。从根本上来说，这就节约了大量的人力物力，这才是真正的"效率第一"。

结构化程序设计方法就是适应这种需要而产生的。

2.8.3 结构化程序的基本结构

结构化程序反对滥用 GOTO 语句而使流程作不规则的跳转。提倡像搭积木一样，由若干基本结构单元顺序组成一个程序。编写程序、阅读程序和执行程序时都是由上到下一个结构一个结构地进行的，也就是程序逻辑顺序与执行顺序一致。

结构化程序规定了三种基本结构，即顺序结构；选择结构；循环结构。并且允许由这三种基本结构派生出其它的基本结构，这些基本结构应当具备以下特性：(1)有一个入口，一个出口(不允许多个出口)；(2)没有死循环(永不终止的循环)；(3)没有死语句(永远没有机会被执行的)或者说：结构中的每一个部分都应当有一条由入口到出口的路径通过它。

三种基本结构如下：

1. 顺序结构。结构内 a 块和 b 块是顺序执行的。见图2.29。

2. 选择结构。根据给定条件是否满足("真"或"假")而选择执行 a 块或 b 块。见图2.30。

3. 循环结构。又分为：

(1) 当型循环结构：当给定条件满足时反复执行 a 块。见图2.31(a)。

(2) 直到型循环结构：反复执行 a 块，直到给定条件满足时才不再执行。见图2.31(b)。

上面提到的"块"可以是一个简单块(例如是一个非 GO TO 语句)，也可以是一个基本结构(即上述三种基本结构之一)。例如图2.32是一个顺序结构，而其中的 b 块又可以是一个选择结构。也就是可用以上三种基本结构组成一个复杂的程序结构。已经证明：任何一个复杂的问题都可以用这三种基本结构组成的程序结构来表示。由以上三种基本结构顺序组成的程序是一个结构化程序。

可以看出，以上三种基本结构都严格遵循"一个入口，一个出口"以及"在这些结构内部任一部分都存在着由入口到出口的一条通道"的原则。除此之外，象图2.33所示的多分支选择结构也符合上述原则，它是由三种基本结构派生出来的另一种基本结构。有了这些基本结

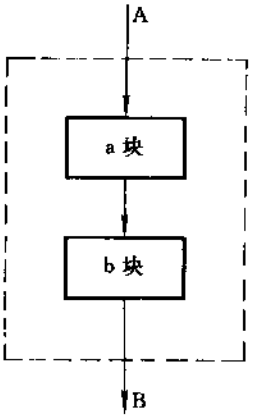


图 2.29

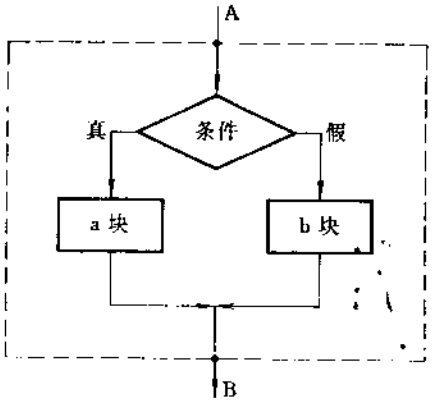


图 2.30

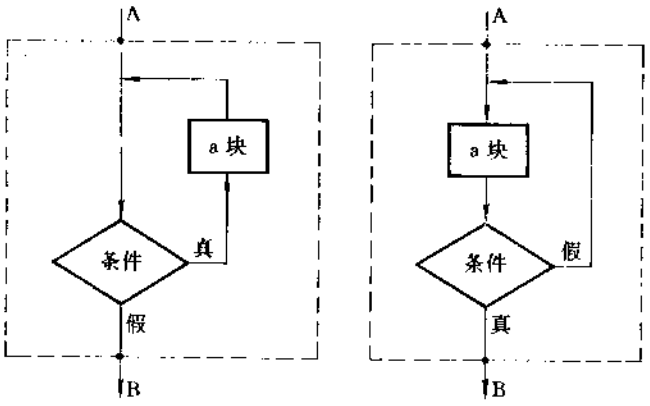


图 2.31

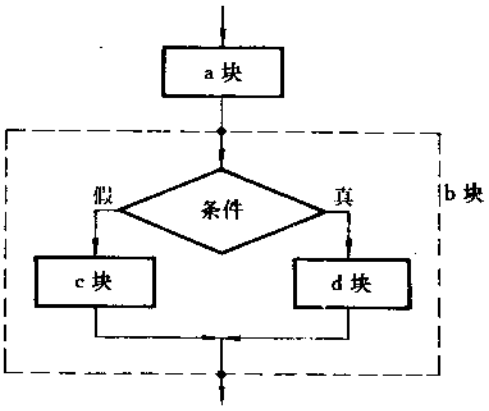


图 2.32

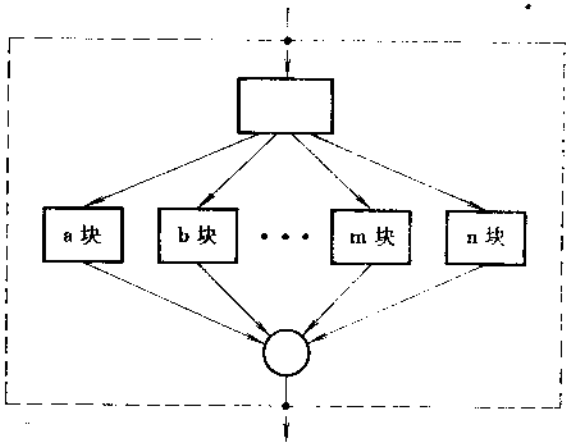


图 2.33

构后我们就可以相对独立地设计程序的各个部分，某一部分的修改决不会影响整个程序其它部分的结构，这种程序结构无疑是比较好的。

2.8.4 结构化流程图

程序设计的一个重要工具是流程图。传统的流程图用流程线指明程序的执行过程，允许用流程线在程序结构中任意转移，它是和过去非结构化程序设计的习惯方法相适应的。我们先看下面的例题：

【例2.10】 假设我国今年的国民总产值为P，并且假定国民经济年增长率为C，C是可变的，对于不同的国民经济年增长率C分别需要多少年(Y)，才能使国民经济总产值倍增。

解 为使国民经济总产值倍增，对于可变的年增长率C，求出相应的年数值Y。现将流程图绘制如图2.34。

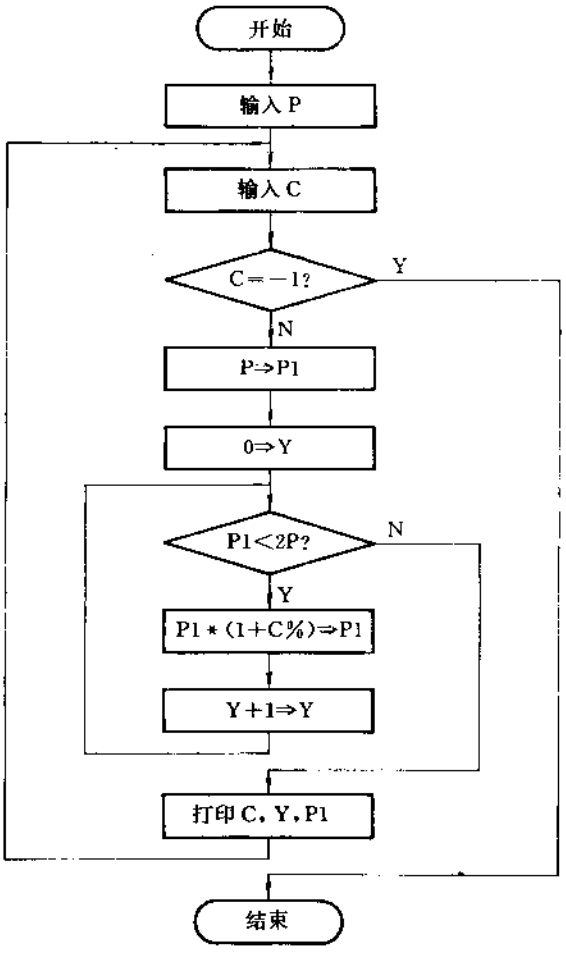


图 2.34

从图2.34可见，此流程图是非结构化的，它并不是由三种基本结构组成的，看这种流程图需要按照箭头指向忽上忽下，很不方便，使人们很难弄清程序的流程。

针对这种情况，在美国 I. Nassi 和 B. Schneiderman 二人1973年提出的方法的基础上，形成了一种适合于结构化程序设计的流程图。这种流程图不同于普通的流程图，它没有流程线和箭头，因此，看流程图时不需要沿着流程线方向上下左右来回跟踪寻找，看结构化流程图就如同看一页书一样，由上而下看下来一目了然。结构化流程图，也称为 N-S 流程图(N 和 S 是以上二位美国人姓名的缩写)，它用下面三种基本成份表示三种基本结构。

(1) 顺序结构。以图2.35形式表示。

(2) 选择结构。以图2.36形式表示。

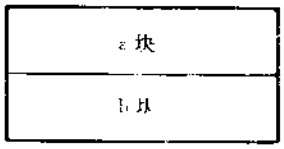


图 2.35

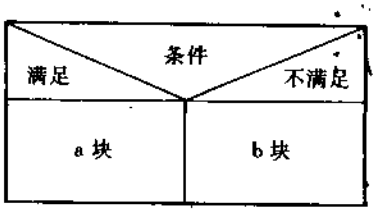


图 2.36

(3) 循环结构。当型循环结构用图2.37(a)形式表示，直到型循环结构用图2.37(b)形式表示。

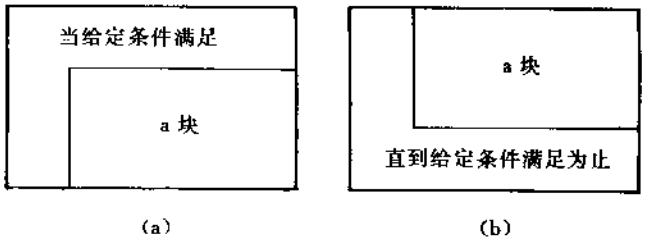


图 2.37

由这种基本成分可以组成一个结构化的流程图。上题解法可以用结构化流程图形式表示。如图2.38所示。

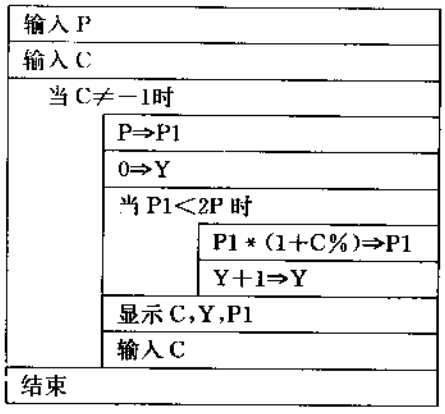


图 2.38

2.8.5 结构化程序设计的实现方法

上面说明了一个结构化程序是由三种基本结构(及由其派生出来的基本结构)组成的，反过来说，一个结构化程序必定能分解为若干个基本结构。但是在拿到一个题目后，怎样着手才能得到一个结构化的程序结构呢?或者说怎样去进行结构化的程序设计呢?我们说，用下面几种方法来实现结构化程序设计。

1. 自顶向下；
2. 逐步细化；
3. 模块化；
4. 结构化编码；

假设有一个总任务，先把它分为若干个子任务，每一个子任务又可再细分为若干个分任务……。见图2.39示意。

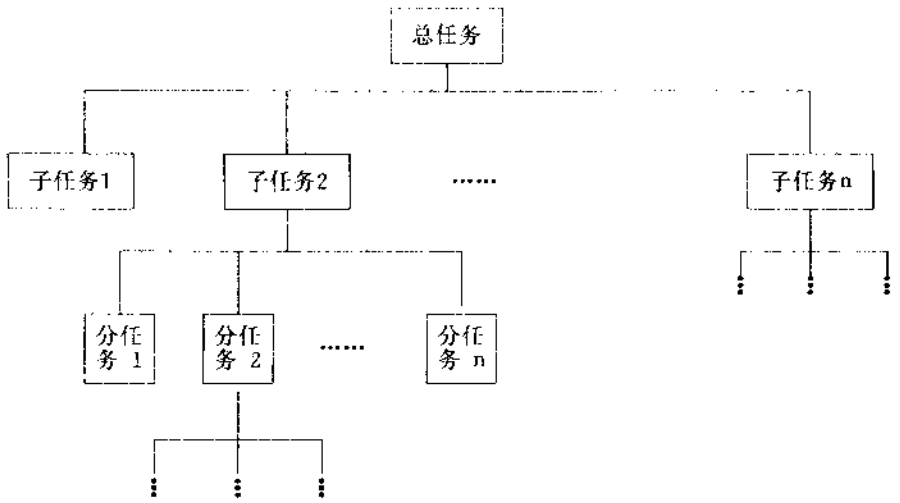


图 2.39

正如同作家想写一本书，先确定一个主题，然后考虑分为若干章，每章下分若干节……，一直细分到足够详细拿起笔就能写出文字为止。这种做法就叫做逐步细化，由于以上的细化是从顶部开始向下逐级进行的，因此，称为自顶向下的逐步细化。这就是由抽象到具体的过程，愈细分就愈具体。可以用一个个模块来实现各项子任务或分任务。一个模块用来实现一个单一的功能(如输入数据，或进行计算，或输出结果等)。在每一模块内又可以按上述方法细化，到最后得到三种基本结构为止，此时就不能再分解了，可据此直接写出程序语句。

根据已设计好的算法(例如以流程图表示的算法)编写的工作称为编码(Coding)。只要算法是正确的，编码工作不应是很困难的。在编码中，要注意使程序清晰，易于阅读。例如采取缩进格式、采用"见名知义"的命名方法等。

如果要解决的任务是一个庞大复杂的任务，那么开发一个软件的步骤还要复杂一些。根据软件工程方法，一个软件从研制到消亡要经历以下几个阶段：

1. 问题定义与需求分析。

2. 总体设计(或称概要设计)。将一个大任务划分为若干子任务,即划分模块。

3. 详细设计。根据子任务(或分任务)的要求,设计算法和数据结构。

4. 编码和单元测试。

5. 综合测试与确认运行。

6. 系统维护。

以上六个阶段称为"软件生存周期"。其中的1.称为"软件定义时期";2.～5.称为"软件开发时期;6.称为"软件维护时期",一般所说的"程序设计"大体包括以上3.,4.两个阶段。一个大任务不是一个人所能完成的,是由系统分析师、系统设计师、主程序员、程序员共同完成的。

关于这方面的知识,本书在此只能作简单的介绍。读者如需作进一步了解,可参阅由谭浩强、傅金铎编著的《结构化程序设计及其在COBOL中的应用》(清华大学出版社出版)。

关于程序设计,可以用一个公式描述:

程序设计=算法+数据结构+程序设计方法+程序环境

也就是说,在进行程序设计时,除了要进行算法和数据结构的设计外,还要运用结构化程序方法,还要了解和运用软件运行的环境(操作系统和程序开发工具),用一种特定的计算机语言去实现它。一个程序人员应当有必要的软硬件知识(尤其是以上的软件知识)才能设计出一个好的软件。

习　题

2.1　按下列题意写出加法语句和减法语句:

(1) 3+A⇒A　(3加A,结果放A中)

(2) 65+A⇒B

(3) 101+A+B⇒B

(4) 9+B+C⇒X

(5) 9+6+7⇒Y

(6) A+B⇒C

　　A+B⇒D

(7) A−B⇒A　(从A中减去B,结果放在A中)

(8) X−15⇒X

(9) X−Y⇒Z

(10) T−10−15⇒U

(11) C−D−E⇒C

　　T−D−E⇒T

(12) H−10−15−16⇒H

　　T−10−15−16⇒T

2.2　写出下面语句执行后的结果。

执行后结果 / 执行的语句 \ 数据项	X (原值50)	Y (原值20)	Z (原值10)
(1) ADD Y TO X			
(2) ADD X, Z TO Y			
(3) ADD 5, 10 GIVING Z			
(4) ADD 10, X GIVING Y			
(5) ADD X, Y GIVING Z			
(6) SUBTRACT Z FROM X			
(7) SUBTRACT Y, Z FROM X			
(8) SUBTRACT 10, Z FROM X			
(9) SUBTRACT Y, 100 FROM 150 GIVING Z			
(10) SUBTRACT Y FROM X GIVING Z			

2.3 按下列题意写出乘法语句和除法语句。

(1) $A\times B\Rightarrow B$

(2) $B\times B\Rightarrow C$

(3) $X\times Y\Rightarrow Z$

(4) $3\times 5\Rightarrow T$

(5) $C\times D\Rightarrow D$

$C\times F\Rightarrow F$

(6) $\frac{y}{x}\Rightarrow Y$

(7) $\frac{150}{T}\Rightarrow Y$

(8) $\frac{N}{K}\Rightarrow J$

(9) $\frac{SUM}{7}\Rightarrow TOTAL$

(10) $\frac{A}{B}\Rightarrow X$

$\frac{A}{B}\Rightarrow Y$

2.4 写出计算语句(COMPUTE 语句),以实现下列的运算。

(1) $T=\frac{A+B}{C+D}$

(2) $S12=[(T^2+5)\cdot(U+4)-3]\div 8$

(3) $X=Y^2+Z^2-Y\cdot Z$

(4) $R=-T+\frac{K}{2}+U\cdot V^2$

(5) $Y3=\frac{Y1+Y2}{A+B}\cdot\frac{C+D}{Y2}$

2.5 假如 Y 是数值型整数数据项,它在内存中的内容原来为:

Y			
8	5	1	2

写出当执行下面的 MOVE 语句后 Y 的内容。

(1) MOVE 12.34 TO Y

(2) MOVE 123 TO Y

(3) MOVE 1 TO Y

(4) MOVE 81245 TO Y

(5) MOVE 87.3 TO Y

(6) MOVE ZERO TO Y

2.6 假设 T 为一组合项，它在内存中的情况为：

T									
T1			T2			T3			
F	U	N	T	A	N	W	A	N	G

写出执行下列 MOVE 语句后 T 的内容。

(1) MOVE 'LIA' TO T1

(2) MOVE 'CHI' TO T1, T2

(3) MOVE 'LI' TO T3

(4) MOVE 'COBOL PRO' TO T

(5) MOVE 'COBOL PROGRAM' TO T

(6) MOVE 'CHINA' TO T3

(7) MOVE 'SHANGHAI'TO T

(8) MOVE 'BEIJING' TO T2

(9) MOVE '186' TO T1

(10) MOVE '2010' TO T

2.7 写一个 COBOL 源程序的过程部分，以实现以下功能：

(1) 用 ACCEPT 语句从计算机指定的设备上输入 T1，T2，T3三个数据。

(2) 求 T1，T2，T3的平均值 T。

(3) 用 DISPLAY 语句显示出 T1，T2，T3和 T 的值。

2.8 按下面流程图(图2.40和图2.41)写出 IF-ELSE 语句。

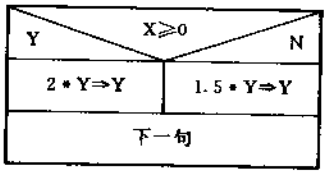


图 2.40

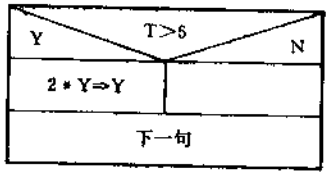


图 2.41

2.9 画出下面句子的流程图和结构化流程图。

```
(1) IF X > 1 MOVE 1 TO T
    ELSE MOVE 0 TO T.
(2) IF X < 100 NEXT SENTENCE
    ELSE MOVE 0 TO Y.
(3) IF W IS GREATER THAN 500
      MULTIPLY 0.9 BY W
    ELSE NEXT SENTENCE.
(4) IF A GREATER THAN B
        MOVE A TO T
        MOVE B TO A
        MOVE T TO B
```

```
        ELSE  MOVE  A  TO  B.
```

2.10　税率规定如下：M＜1000元的为5%，1000≤M＜1500的，为10%。1500≤M＜2000的，15%。M≥2000元的，为20%。M 为需计算税率的金额。请编程序的过程部，求应交税款（只要求到整数位）。由打印机输出应交税款（需要定义输出文件）。

2.11　将例2.9改写成完整的 COBOL 程序。

2.12　由磁盘机读入一个班50个学生的姓名，每读入一个记录，即将该学生的姓名打印在纸上，每打印完一行后空一行。然后再打印第二个学生的名字。

程序的前三部分已给出，请写出过程部分。输入文件名为 INPUT-FILE，输出文件名为 OUTPUT-FILE。

```
IDENTIFICATION DIVISION.          (标识符)
PROGRAM-ID.  EX.
ENVIRONMENT  DIVISION.            (环境部)
INPUT OUTPUT  SECTION.
FILE-CONTROL.
  SELECT  INPUT FILE  ASSIGN TO U01.
                      (U01为外部文件名字)
  SELECT OUTPUT FILE ASSIGN TO U02.
                      (U02为外部文件名)
DATA DIVISION.            (数据部)
FILE SECTION.
FD  INPUT-FILE  LABEL  RECORD  IS  STANDARD.
01  NAME PIC X(80).      (输入文件的记录名)
FD  OUTPUT FILE LABEL RECORD IS OMIT TED.
01  OUTPUT-REOORD.       (输出文件的记录名)
    02  FILLER PIC X.
    02  OUTLINE  PIC X(132).
```

第三章　标识部和环境部

§3.1　标识部(IDENTIFICATION DIVISION)

COBOL 源程序中的第一部分是标识部。它是程序中不可缺少的部分。标识部的作用是标识一个 COBOL 源程序，即为程序加上标志，以便识别。COBOL 规定，每一个程序(包括主程序或子程序)都必须有一个名字，当系统需要找某一个程序时，就按名字查找。即按名字管理和调用。此外，程序员还可以在标识部中填入写程序的日期、源程序编译的日期以及其它信息，这些只是为程序员作备忘之用。因此，标识部可以分为必写的部分和可任选的部分。

3.1.1　标识部的必写部分

```
IDENTIFICATION  DIVISION.
PROGRAM ID.  程序名.
```

第一行是"部头"。第二行为程序标识段，其中 ID 是 Identification(标识)的缩写，ID 后应有句点，并空一格，然后写上由程序编写者定义的名字。有的系统对程序名作了一定的限制，如不能多于 6 个字符，且只限于字母和数字(不包括连接符)。有的则规定不超过 30 个字符。

程序标识段的作用是给程序起一个名字，这部分是必要的，而且必须写作第一段，即紧接在"部头"的下面一行。

标识部下面不设节，只设段。"部头"和"段头"都从 A 区开始书写。

3.1.2　标识部的任选部分

除程序标识段外，标识部可以没有其它段。但允许程序编写者在这部分中写入某些信息以备忘。例如作者、日期、保密程度等等。这些信息并非程序的必要部分，对程序的执行不产生任何影响，只是为看程序时起个"说明"作用。也就是说它是写给人看的，不是为计算机执行时用的，仅仅是一个"备忘录"。

它的一般格式为：

```
[AUTHOR.                作者姓名.]
[INSTALLATION.          计算机设置的场所.]
[DATE-WRITTEN.          源程序编写的日期.]
[DATE-COMPILED.         源程序编译的日期.]
[SECURITY.              保密程度.]
```

以上六段可有可无，也可以选写其中几段，次序任意。初学者完全可以不写它们，以便程序简洁些。

下面举例说明标识部的用法：

```
IDENTIFICATION DIVISION.                    (标识部头)
PROGRAM-ID.  EX.                            (程序名为 EX)
AUTHOR. ZHANG-LI.                           (作者：张利)
INSTALLATION. TSINGHUA UNIVERSITY.          (编写的场所：清华大学)
DATE-WRITTEN. 1992/5/30.                    (编写日期：1992 年 5 月 30 日)
DATE COMPILED. 1992/6/5.                    (编译日期：1992 年 6 月 5 日)
SECURITY. THIS PROGRAM RESTRICTED TO PERSONNEL WHO HAVE
        BEEN CLEARED BY THE CONTROLLER'S OFFICE.
        (保密：此程序仅限于已被检查人员审查通过的人使用)
```

读者通过此例，完全可以了解各段的作用和用法。

§3.2 环境部(ENVIRONMENT DIVISION)

环境部的作用是说明程序运行的环境，即程序是在什么软硬件环境下运行的。这是整个 COBOL 程序中唯一与计算机硬件设备有关的部分。在这部分中要说明程序用到哪些设备，哪些文件，将程序中用到的内部文件名与外部文件(或外部设备)之间建立起联系。

前面已说过，COBOL 将外部设备和外部文件只集中出现在环境部，是为了提高程序的可移植性，不致因外部环境的改变而导致程序中每一部分都需作相应的修改，而只需修改环境部即可。

3.2.1 环境部的一般格式

```
ENVIRONMENT DIVISION.                              (环境部)
CONFIGURATION SECTION.                             (配置节)
SOURCE COMPUTER.  源计算机名.
OBJECT COMPUTER.  目标计算机名.
[SPECIAL NAMES.  专用名描述项]
[INPUT-OUTPUT SECTION.                             (输入输出节)
FILE-CONTROL.  {文件控制描述体}…
[I-O-CONTROL.  输入输出控制描述体]]
```

可以看出：

(一) 环境部包括两个节：

环境部 { 配置节 (CONFIGURATION SECTION)
　　 { 输入输出节 (INPUT-OUTPUT SECTION)

(二) COBOL 要求，配置节(CONFIGURATION SECTION)是必须写的。但不少计算机系统允许可以不必写配置节(包括节头和源计算机名段以及目标计算机段)。因此在本书中为使程序简练些，往往把这一部分省略了。

(三) 输入输出节按规定是任选的，但只有最简单的程序(用 ACCEPT 语句和 DISPLAY 语句进行输入和输出少量数据)才不用输入输出节。只要使用文件进行输入或输出的，都应在环境部中写出输入输出节。

下面分别介绍环境部中各节的内容

3.2.2 配置节（CONFIGURATION SECTION）

（一）本节包括三段：源计算机段、目标计算机段、专用名段。下面为这节的具体用法：

```
CONFIGURATION          SECTION.                    (配置节)
SOURCE-COMPUTER.       IBMPC-386.                  (源计算机段)
OBJECT-COMPUTER.       IBMPC-386.                  (目标计算机段)
        MEMORY  SIZE  IS  5000  WORDS.
SPECIAL-NAMES.                                     (专用名段)
        CONSOLE  IS  A1.
```

它说明源程序是在 IBMPC-386 计算机上编译的。源计算机指的是编译源程序时使用的计算机。目标计算机指的是经过编译后所得到的目标程序运行时所使用的计算机，即数据处理阶段所使用的计算机，本例中目标计算机也是 IBMPC-386。这二者可以是同一计算机，也可以是不同的计算机。如果是不同的计算机，则应要求这二台计算机是兼容的。因为一种型号的计算机编译出的目标程序，在另一型号的计算机上往往是不能运行的。

在配置节中还应说明程序运行时需要目标计算机提供的最小容量。此例中说明该目标程序运行时将要占用内存大约 5000 字左右。如果目标计算机所提供的能使用的内存容量小于此数字，则程序不能正常运行。

第五、六行为“专用名段”，说明 CONSOLE（某计算机系统指定的“控制台”的专用名）在本程序中可以用 A1（助忆名）来代表。

（二）源计算机段和目标计算机段的一般格式

```
SOURCE-COMPUTER.    源计算机名.
OBJECT-COMPUTER.    目标计算机名.
[ MEMORY  SIZE IS  整数 { WORDS      } ]
                        { CHARACTERS }
                        { MODULES    }
```

对内存大小的说明，可以用 CHARACTERS（字符），或 WORDS（字），或 MODULES（模块）之一。

（三）专用名段。专用名段用来通知系统把系统中原规定的一些设备名或功能名或符号改为用户自己指定的名字或符号。计算机对有关的硬设备和某些功能（如打印时不换行，跳到本页末或下页头等），都赋予一个特定的专用名。例如有的系统把打印机定名为 PRINTER，把控制台定名为 CONSOLE，有的把宽行打印机定为 SYSOUT，把读卡机定为 SYSIN 等（请注意，每个系统所起的专用名是不同的，不可照搬，使用时应弄清楚本系统各硬设备的专用名是什么）。程序编写者可以用自己规定的助忆名去代替系统的专用名。如果程序中不规定自己定的助忆名，则此段可不写。

欧洲人的习惯是用逗号代表小数点，如 12.34，他们写成 12,34。对此 COBOL 提供一项说明，可在专用名段中写：

```
DECIMAL-POINT  IS  COMMA.  (小数点是逗号)
```

这样，凡程序中数值内出现的小数点，在输出时一律会自动改成逗号。

又如，美国用“$”作货币符号，而其它国家可以改用其它符号，如用“L”表示英磅，用

"Y"表示人民币等。可在专用名段写：

CURRENCY SIGN IS 非数值常量

如：CURRENCY SIGN IS 'L'.

则在出现货币符号时，一律会自动以"L"代替"$"。但不能用0～9,A,B,C,D,E,P,R,S,V,X,Z字符作货币号。

专用名段的一般格式(在配置段中)

```
SPECIAL-NAMES.
    [DECIMAL-POINT  IS  COMMA.]
    [CURRENCY  SIGN  IS  非数值常量.]
    [专用名  IS  助忆名.]
```

使用助忆名代替系统固有的专用名，纯粹是为了便于记忆。因为在有些系统中专用名不好记忆，程序编制者希望以简单易记的名字代替它。此外，当改用其它计算机系统时，只需改变专用名段的专用名即可，程序中过程部的所有语句都不必改动。

3.2.3 输入输出节（INPUT-OUTPUT SECTION）

(一) 程序中如果用到输入输出文件，就应在这个节中把程序中的内部文件与外部文件(包括外部设备)联系起来。

输入输出节包括两个段：

输入输出节 { ① 文件控制段（FILE-CONTROL.）
② 输入输出控制段（I-O-CONTROL.）

文件控制段是为文件分配外部文件。只要用到INPUT-OUTPUT节，文件控制段就是必写的。输入输出控制段的作用是可以指定目标程序运行时，几个文件共用一个内存区，以省内存。初学者不会用到这部分。在此不作介绍。

(二) 文件控制段是环境部的重要部分。它的主要功能是给程序中使用的文件命名。指出存放该文件的外部设备及其它有关文件控制的信息。程序中只要用到了文件，则必须有一个对该文件的描述部分，它是由SELECT子句实现的。文件控制段的一般格式

```
INPUT-OUTPUT  SECTION.        (输入输出节)
FILE CONTROL.                 (文件控制段)
SELECT 文件名  ASSIGN TO      外部文件名.
```

(三) 说明

1. 紧跟SELECT后面定义的文件名是指源程序中用到的内部数据文件名(在数据部中定义)，COBOL程序中的READ,OPEN等语句都是用该文件名进行操作的。

2. 在本书第二章§2.2中的2.2.3段(READ语句)中我们已说明了在环境部中用SELECT子句将内部文件名与外部文件名联系起来。外部设备也视作一种外部文件。但应注意，在不同的计算机系统中，如何表示"外部文件名"的方法是不同的，大体有以下三种方法：

(1) 在SELECT子句的"ASSIGN TO"的后面写上磁盘上实际的文件名。例如在小型机VAX系列机上的用法如下：

```
FILE-CONTROL.
    SELECT  FILE1  ASSIGN  TO  DUA1:COB.FILE1.DAT
```

内部文件名为"FILE1"，外部文件名为"DUA1：COB. FILE1. DAT"，它表示在磁盘DUA1的COB子目录下的FILE1. DAT文件。在使用磁盘文件时应当了解"目录"的概念。在使用磁盘时，为了使用和管理的方便，往往将磁盘分为若干部分，每一个用户只能在指定的部分内存储和操作自己的文件，例如规定A1用户使用的部分为A1区，A2用户用A2区…。*这样做可以防止用户误操作不属于他的文件，例如A1不能使用A2的文件。这种划分*的各部分便称为"子目录"，例如A1子目录，A2子目录…。在A1子目录下的FILE. DAT文件就表示为A1. FILE. DAT(在IBM-PC的DOS操作系统下，表示为A1\FILE. DAT)。子目录下还可以有子目录，例如在A1子目录下又可以设立A11子目录，A12子目录…等，在A11子目录下的FILE. DAT文件表示为A1. A11. FILE. DAT(在DOS操作系统下表示为A1\A11\FILE. DAT)。这种管理方法如同一本书的目录一样，分为章、节、段，一层一层，互相区别开，便于查阅，所以称为"目录"。有些文件不放在子目录下而直接处于"根目录"管理下，调用文件时就不必写出子目录名，例如只需写"DUA1:FILE1. DAT"即可。

小型机多为多用户系统，在使用计算机时，系统管理员已为每一个用户分配了所用磁盘空间和子目录，用户的源程序和所用的数据文件一般都放在这个子目录中。调用本子目录中的文件就不必写子目录名。例如在子目录COB下运行的程序需要从这个子目录中的FILE. DAT中读入文件，可以不必写子目录名"COB"。例如下面两行等价：

```
SELECT  FILE1  ASSIGN  TO  DUA1:COB.FILE.DAT.
SELECT  FILE1  ASSIGN  TO  DUA1:FILE.DAT.
```

但如果要从另一个子目录A1下的FILE. DAT文件读入数据，则必须写出该子目录名：如：

```
SELECT  FILE1  ASSIGN  TO  DUA1:A1.FILE.DAT.
```

数据文件的后缀一般用"DAT"。系统规定，当用"DAT"作数据文件的后缀(扩展名)时，".DAT"可以省略不写。例如，可写成：

```
SELECT  FILE1  ASSIGN  TO  DAU1:FILE.
```

也就是说，当文件名不带后缀"DAT"时，系统会自动加上".DAT"而找寻"FILE. DAT"文件。

(2) 在SELECT子句只指出外部设备名。例如在IBM-PC微机系统中，以PRINTER代表打印机，以DISK代表磁盘，因此其输入输出节中的文件控制段的格式如下：

```
FILE-CONTROL.
    SELECT  IN-FILE  ASSIGN  TO  DISK.
    SELECT  PRINT-FILE  ASSIGN  TO  PRINTER.
```

其中IN-FILE和PRINT-FILE是内部文件名，IN-FILE是一个磁盘文件，PRINT-FILE是一个打印文件。但IN-FILE代表磁盘上哪一个文件呢？在这里并未解决，是在数据部中解决的。在数据部的"文件节"中的FD描述体中用VALUE OF子句来指出该文件的实际文件名。例如：

```
DATA  DIVISION.          (数据部)
FILE  SECTION.           (文件节)
FD IN-FILE LABEL RECORD  STANDARD
    VALUE OF FILE-ID "C:INF.DAT".
```

关于数据部中的内容，将在第四章作详细说明。在这里只是由于要说明内部文件名与外部文件名的关系才顺便介绍有关数据部的内容。在数据部中的文件节要描述文件的属性，FD 描述体说明文件 IN-FILE 的“标号记录是标准的”(所有磁盘文件都应如此写明)，VALUE OF 子句用来指出：IN-FILE(内部文件名)代表“C:INF.DAT”，即 C 盘上的 INT.DAT 文件。

(3) 在一些中、大型计算机系统，在 SELECT 子句中用该系统指定的逻辑名作为外部文件名，然后用作业控制语句将该逻辑名与实际的设备或文件相联系。例如在中型机 M-150 中，环境部输入输出节中的文件控制段用以下形式：

```
FILE-CONTROL.
  SELECT IN-FILE
         ASSIGN TO SYS010-CR.
  SELECT DA-FILE
         ASSIGN TO SYS012-DA-S-DISK.
  SELECT PRINT-FILE
         ASSIGN TO SYS013-LP.
```

内部文件分别是 IN-FILE，DA-FILE 和 PRINT-FILE. ASSIGN TO 后面是文件的逻辑名。把 IN-FILE 文件分配给 SYS010-CR(读卡机)，DA-FILE 文件分配给 SYS012-DA-S-DISK (磁盘机)，PRINT-FILE 文件分配给 SYS013-LP(宽行打印机)。以上 SYS010，SYS012…中的“SYS”是必写的，010，012，…则是用户自定的，也可以指定为其它数字。SYS012-DA-S-DISK 中的 DA 表示磁盘文件，“S”表示“顺序文件”，它们说明了文件的类型。在程序外面，用操作系统提供的“作业控制语句”将上述逻辑文件名与实际的设备(文件)相联系。如：

```
ASSGN  SYS010, SYSRDR  (读卡机)
ASSGN  SYS012, DISK, VOL=OS4F22,SHR  (磁盘机,卷号为OS4F22,共享方式)
ASSGN  SYS013,SYSLST  (宽行打印机)
```

上面的 SYSRDR 是 M-150 系统指定的读卡机的专用名，ASSGN 命令的作用是将程序中指定的 SYS010 分配给读卡机，将 SYS012 分配给卷号为 OS4F22 的磁盘机，该文件是共享方式。将 SYS013 分配给宽行打印机。

关于以上三种方式，只作简单介绍，以便读者有所了解，在遇到不同的系统时不致感到茫然。至于每一种系统中的用法，可详见所用系统的使用手册，或请教机房工作人员。

最后把标识部和环境部集中举一例介绍一下：

```
IDENTIFICATION    DIVISION.
PROGRAM-ID.  EXAM.
AUTHOR.      WANG-LI.
DATE-WRITTEN.    1992/7/1.
DATE-COMPILED.   1992/8/1.
ENVIRONMENT      DIVISION.
CONGIGURATION    SECTION.
SOURCE-COMPUTER.   IBMPC 386.
OBJECT-COMPUTER.   IBMPC 386.
SPECIAL NAMES.
CURRENCY  IS  'Y'.
INPUT-OUTPUT    SECTION.
FILE-CONTROL.
     SELECT  A1  ASSIGN  TO  DISK.
```

SELECT A2 ASSIGN TO PRINTER.

习　题

3.1　标识部唯一的必写部分是什么？

3.2　环境部的作用是什么？

3.3　在环境部中集中说明计算机的硬件环境有什么好处？

3.4　了解你所用的计算机系统是如何使用外部文件名的。

3.5　请写出一个程序的标识部和环境部。

第四章　数据部之一

§4.1　概　　述

4.1.1　数据部的作用

数据部(DATA DIVISION)是整个 COBOL 源程序中唯一描述数据的部分。它是四个部分中的第三部分，是任何一个源程序中不可缺少的部分。

凡在程序中涉及的全部数据(包括输入的数据、输出的数据、中间数据)都要在数据部中加以说明(描述)。

数据有两种：(1) 一种是孤立的数据项。如 A 和 B 是两个互相独立、没有内在联系的数据项，它们各自占据内存中的一个域，是具有独立逻辑意义的项，它们就称为孤立数据项。

(2) 组合的数据项。数据是互相关联的，它们之间是存在内在联系的，如：

STUDENT-SCORE (学生成绩记录)					
NAME (名字)	NUM (学号)	CORUSE SCORE (课程成绩)			
		CORUSE－1 (课程 1)	CORUSE－2 (课程 2)	CORUSE－3 (课程 3)	AVERAGE (平均)

学生成绩记录(STUDENT-SCORE)包括若干项，其中课程成绩(CORUSE SCORE)又包括几项。这些数据间是存在一定的内在联系的，它们都属于同一个学生的成绩记录。而它们之间又有一定的从属关系。在这些数据中，有些项是数值型的(如成绩)，有些项是非数值型的(如名字)。

程序过程部中出现的所有数据项都应当在数据部中对它们的属性进行说明。包括：

(1) 每一个数据项的类型(是数值型或字符型…)，它们在内存中的存储形式，它们的长度(占多少字节)。

(2) 数据项间的相互关系。有的数据项之间有从属关系，例如一个组合项包括若干个初等项。哪些数据项是文件记录中的一部分？哪些数据项与文件无关，即不从属于记录。

(3) 描述记录与文件的关系，即内存中的输入输出记录区是与哪一个文件有关的？

(4) 文件的属性。由多个记录组成一个“物理块”，一个记录包含多少个字节？文件有无标号记录等。

总之，数据部的作用是定义数据项属性，描述数据结构，准备被加工的“原料”。而过程部则是对这些“原料”进行加工。COBOL 把数据部和过程部分开，使每一部分的任务都比较单纯、清晰，便于书写和修改。例如想修改输出数据格式而不修改操作，则只修改数据部即可，与过程部无关。

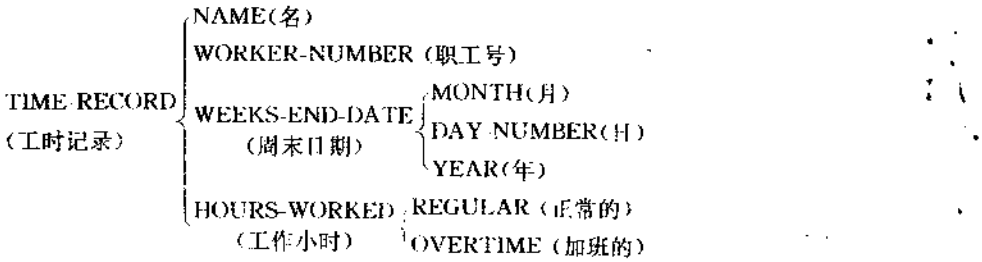
4.1.2　数据的层次和层号

我们已在第一章中简略地介绍了数据层次结构的概念，并在前几章中的例子中使用过

它们。在这里我们再作系统的介绍。

COBOL 中把有从属关系的数据用层次(level)关系来描述。数据的层次结构是:记录→组合项→初等项。

逻辑上不能再细分的项,称为初等项(亦称基本项,Elementary item)。包含若干个初等项的项称组合项(Group item)。数据项的最高层次是记录(Record)。在一个记录中可以分为若干层次。如有一"工人的每周工时记录",其数据关系如下:



TIME-RECORD 是记录名,WEEKS-END-DATE 和 HOURS-WORKED 是组合项。我们可以在数据部中用以下层次关系来表示:

```
01  TIME RECORD.
  03  NAME PIC  X(20).
  03  WORKER-NUMBER  PIC  9(6).
  03   WEEKS-END-DATE.
    05  MONTH  PIC  99.
    05  DAY NUMBER  PIC  99.
    05  YEAR  PIC  99.
  03   HOURS WORKED.
    06  REGULAR     PIC  99.
    06  OVERTIME    PIC  99.
```

层次规定如下:

(1) 用来描述数据的层次结构的层号从 01 开始,到 49,即可用的层号为 01～49。记录的层次最高,定为 01 层号,如上例中,TIME-RECORD 为记录名,层号 01。没有比 01 更高的层次。

(2) 从属项的层号比其上属项的层号高。即层号小的组合项包含层号大的数据项(组合项或初等项)。如 03 层从属于 01 层,05 层(或 06 层)又从属于 03 层。

层号不必要求连续。如果把上例的 03 层改为 02 或 04,作用相同。换句话说,如果 B 与 C 为 A 的从属项,则 A 的层号应小于 B,C。

```
02  A.
    03  B…
    03  C…
```

(3) 如果几个数据项都从属于同一组合项但互不从属,则这几个数据项应具有相同的层号,如上面表示的那样。而如果写成:

```
02  A.
    03  B…
    04  C…
```

则 B 和 C 不属同层,C 又从属于 B。

在上面的例子中,MONTH 和 REGULAR 虽用了不同的层号(05 和 06),但它们属同

一层次，它们都是同一层号(03层)的组合项下面的从属项。如果把05改成07也一样。这是因为WEEKS-END-DATE和HOURS-WORKED是两个不同的组合项。只要求每一个从属项的层号分别大于所在的组合项的层号即可。而不要求不同的组合项下面从属项层号都相同。

一个层号为K的组合项包括它下面所有的层号比它大的组合项和初等项，直到遇到层号小于K或等于K的层次为止。如上例中WEEKS-END-DATE组合项层号为03，它的范围包括下属的05层(如果05层还有从属项06，07…的话，也都包括在05层之内)，直到遇到另一个03层HOURS-WORKED为止。

4.1.3 数据部的结构

数据部中通常用到的有以下几个节：

(一) 文件节(FILE SECTION)

用来描述程序中用到的输入文件和输出文件及其记录中各数据项的属性。

(二) 工作单元节 (WORKING-STORAGE SECTION)

用来描述程序中用到的中间数据项。

(三) 联接节 (LINKAGE SECTION)

用来描述与调用程序间发生数据传递的数据项。

(四) 报表节 (REPORT SECTION)

为了完成报表编制功能，此节用来规定欲输出的报表的"体裁"，设计各报表栏的打印形式和方法等。

本章中主要介绍文件节和工作单元节，其它节将在本书下册有关章节介绍。

§4.2 文件节 (FILE SECTION)

4.2.1 文件节的作用

程序中每一个输入或输出文件都要在文件节中加以描述。描述的内容包括：

(1) 文件名和文件属性。

(2) 文件中包括的记录的名字。

(3) 每个记录中数据的层次关系。

(4) 记录中各数据项的数据形式和占内存的大小。

文件节应用举例：

```
DATA  DIVISION.          (数据部)
FILE SECTION.            (文件节)
FD   ACCOUNTS-RECEIVABLE    (这是文件名)
     LABEL RECORD  IS  STANDARD.
     DATA  RECORD  IS  RECEIVABLE.
01   RECEIVABLE.      (这是记录名)
   02  ACCOUNT  RIC  9(6).
   02  FILLER   RIC  X(3).
   02  AMOUNT   PIC  9(6).
   02  NAME     PIC  X(65).
```

4.2.2 文件描述

从上面可以看到，文件描述体不是用层号开始，而用 FD 开始。在第二章中已说明：FD 是 File Description（文件描述）的缩写。FD 称为文件描述符或层指示符。FD 后面跟文件名。上面的例中的文件名为 ACCOUNTS-RECEIVABLE(是由程序员自选的)。它是程序中用到的内部文件名，是在环境部中定义过的。

LABEL RECORD IS STANDARD 表示"标号记录是标准"的。只有磁盘(带)文件才有标号记录，而且一律定义为标准的标号记录，因此必须写成"LABEL RECORD IS STANDARD"。而卡片文件，打印文件等是没有这种"标号记录"的。因此应写成："LABEL RECORD IS OMITTED"，即"标号记录省略"。

DATA RECORD IS RECEIVABLE. 表示文件中包含记录名为 RECEIVABLE 的数据记录。这一项是可以省写的。

最简单的文件描述体的一般格式为：

```
FD    文件名
    LABEL {RECORD IS  } {STANDARD}
          {RECORDS ARE} {OMITTED }
    [DATA {RECORD IS  } 数据名]
          {RECORDS ARE}
```

文件描述还有其它内容。待以后用到时再介绍。其它一些用得较少的可选项，读者可在今后用到时自己查说明书即可。

4.2.3 记录描述

记录描述体由 01 层号开头，后跟记录名。如上例中的：

```
01  RECEIVABLE.
```

注意本行末尾有一句点和空格。

上面例中，记录 RECEIVABLE 包括四个初等项：ACCOUNT（帐号），FILLER（填充项名），AMOUNT（金额），NAME（名字）。在数据部的文件节中要描述出记录的层次关系以及描述每个初等数据项的类型和长度。

如果记录下面不再分项，即记录本身就是一个初等项，则这种记录描述体最简单。如：

```
01  RECEIVABLE  PIC  X(80).
```

它表示 RECEIVABLE 记录是一个初等项，字符型，在内存占 80 字节。如果记录下又包括若干项，则应按下面的介绍对每一个数据项逐项进行描述。

4.2.4 数据项描述

在每一个初等项(上例中 02 层)的名字后跟一个 PIC 子句。用它来描述数据的类型和长度。例如上例中：

```
ACCOUNT   数值型，长度为 6 个字节。
FILLER    字符型，长度为 3 个字节。
```

AMOUNT　数值型，长度为 6 个字节。
NAME　字符型，长度为 65 个字节。

关于 PIC 子句，在下一节中再作详细介绍。数据项描述体还有其它内容，将逐步介绍。

4.2.5 文件节的书写格式

FD 从 A 区开始书写，01 层号也从 A 区开始，其它层号（如 02）可以从 A 区也可以从 B 区开始。为了看起来层次清楚，最好从 B 区开始，并按层次关系写成锯齿形状。如：

```
01  A.
    02  B.
      03  C…
```

4.2.6 举例

为了说明文件节在程序中的作用以及程序中几部分的联系。我们举一个简单的然而是完整的程序例子（这个例子在程序纸上的书写格式见图 1.6）。

【例 4.1】

```
IDENTIFICATION        DIVISION.
PROGRAM-ID.      EXAM-41.
ENVIRONMENT           DIVISION.
INPUT-OUTPUT          SECTION.
FILE-CONTROL.
    SELECT  ABC  ASSIGN  TO  FILE1.
    SELECT  XYZ  ASSIGN  TO  FILE2.
DATA              DIVISION.
FILE  SECTION.
FD  ABC  LABEL  RECORD  IS  STANDARD.
01  A  PIC  X(80).
FD  XYZ  LABEL  RECORD  IS  STANDARD.
01  B.
    02  FILLER  PIC  X.
    02  C  PIC  X(80).
PROCEDURE         DIVISION.
K. OPEN  INPUT  ABC  OUTPUT  XYZ.
T. READ  ABC  AT  END  GO  TO  J.
   MOVE  A  TO  C.
   WRITE  B  AFTER  ADVANCING  2  LINES.
   GO  TO  T.
J. DISPLAY  'PROGRAM  END'.
   CLOSE  ABC, XYZ.
   STOP  RUN.
```

假设我们所用的计算机系统所用外部文件名的方法是：在 SELECT 子句中的 ASSIGN TO 后面直接写磁盘文件名（后缀"DAT"可省略），则本程序的作用是从磁盘文件 FILE1.DAT 中读取记录，并写到磁盘文件 FILE2.DAT 中去。当遇到文件结束标志时，显示出："PROGRAM END"字样，程序停止运行。

标识部给出程序名 EXAM-41。环境部描述了该程序运行时的环境。规定内部文件名 ABC 与磁盘上的外部文件名 FILE1.DAT 联系，内部文件名 XYZ 与磁盘上的外部文件名 FILE2.DAT 联系。

在数据部中对文件 ABC 和 XYZ 分别进行描述。文件 ABC 中的记录名为 A，它是一个初等项，PIC 子句中的"X"表示字符型，长度为 80 个字节。文件 XYZ 中记录名为 B，它包含两个初等项。FILLER 占一个字节，它的作用是控制走纸（所谓走纸控制，不仅只是在打印机输出起作用，在向磁盘文件写记录时，同样起作用。例如记录的第一个字符将被吃掉，控制记录之间是否有空记录等）。

在过程部中，先打开输入文件 ABC 和输出文件 XYZ。由于在环境部中已把内部文件和外部文件相联系。READ ABC 表示从磁盘文件 FILE1.DAT 中读入一个记录到内存的输入记录区，该输入记录区是在数据部中定义 ABC 文件时定义其记录名为 A 的，长度为 80 字节，数据类型为字符型。在读入一个有效记录后，就放在记录区 A 中。执行 MOVE A TO C 时，把 A 的内容传送给数据项 C，而数据部中已定义了数据项 C 是数据项 B 的一个从属项。因此数据项 B 中也已放有数据了。在执行"WRITE B"语句时，由于 B 已被定义为文件 XYZ 的输出记录区的名字，因此从内存输出 B 就必然输出到文件 XYZ 中。由于在环境部中已将内部文件名 XYZ 与外部文件 FILE2 相联系，所以就是把记录 B 中的数据输出到文件 FILE2 上。"GO TO T"语句的作用是构成循环，不断重复上述过程（从文件 ABC 读入一个记录，再传送到 B 中，然后把它输出到 FILE2 文件中），直到读完文件中全部记录，然后转到 J 段，显示"PROGRAM END"，关闭所有文件，结束运行。

数据项 B 中，只在数据项 C 中存放了数据，在 FILLER 中并未送入数据。由于输出记录中的第一个字符是被系统"取走"作走纸控制符用的，它的内容是什么是无关紧要的。所以不必向它传送任何数据。由于这个数据项只起"占一个字符位置"的作用，而不会也不可能输出它的内容，因此用保留字 FILLER 作数据项名。其实也可以不用 FILLER 而用其它用户定义的名字（如 X，Y，Z 等）也是可以的。但一般习惯上都用 FILLER，以表示与其它有用的数据项相区别。

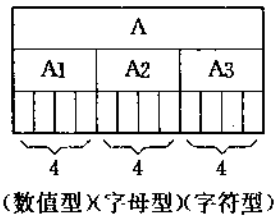
可以看到，在整个程序中，文件名 ABC 共出现五次。过程部中出现三次（打开，读，关闭），这是过程部的要求，它要求执行相应的动作。环境部中的文件控制段使内部文件名和外部文件名相联系。数据部中文件节对输入和输出文件进行描述。建议读者通过这个完整的例子弄清楚程序各部分的作用和它们之间的联系。

§4.3 字型子句（PIC 子句）

PIC 是 Picture 的缩写。PIC 子句用来描述每一个初等数据项。它说明：(1) 数据项是什么类型的（数值型？字符型？字母型？…），如果是数值型的，是否包含正负号和小数点。(2) 数据项占多大的内存域。(3) 是否需要按打印的要求准备有关字符（如打印出 $，+，-，* 等符号）。

例如：

```
01   A.
     02   A1   PIC   9999.
     02   A2   PIC   AAAA.
     02   A3   PIC   XXXX.
```



A 是记录名，下属三个初等项。A1 占内存 4 个字节，

每个字节中可以放一个数字，以 9 表示在该位置上可放入一个 0～9 中的一个数字。四个 9 表示可以放四个 0～9 之间的数字。也可以写成 9(4)。A2 中可以放四个字母，也可以写成 A(4)。A3 中可以放四个字符，也可以写成 X(4)。

PIC 子句的主要作用在于描述一个初等项的一般特征(类型、长度)和编辑要求。

4.3.1 数值型数据的描述

(一) "9"描述符

表示在该位置上可以放入一个 0～9 之间的数字，因此：

```
01  A  PIC  999.
```

表示 A 可以放入 000～999 之间的一个三位数，如 134,256,791 等。有几个 9 表示可以放入几位数字。例：

描　述	数值	在内存中表示
02 X PIC 9999.	1234	1234
02 Y PIC 9(5).	467	00467
02 Z PIC 99.	86	86
02 T PIC 9(6)	11011	011011

注意：(1) 在数值型数据项中只能放入 0～9 之间的数字，而不能放入空格。如果数据项 E 用 PIC 9(6)描述，想从键盘输入数值 1 2 3 给 E，不应该用"␣␣␣ 1 2 3"，而应是"000123"(不包括引号)，因为空格不是数字。

(2) 用"9"描述符只能表示整数。如果输入小数部分，则小数部分被舍去。

```
MOVE 1.2 5 TO B
```

如果 B 的描述为 PIC 9。则 B 中内容为 1(见右图)。

B
1

(3) 如送入一个负数，则负号被舍去。

```
MOVE −1.25  TO  B
```

B 中内容仍然是 1。即只存入数值的绝对值。

如想存入正负符号，则应使用后面介绍的 S 描述符。

(二) "V"描述符

指出在数值数据结构中隐含的小数点的位置。如：

```
03  M  PIC  999V99.
```

表示 M 在内存中占 5 个字节，可放五个 0～9 数字，指出前三个数字与后两个数字间有一隐含的小数点。(此小数点不占内存单元。它是由编译系统置一信息，表示在该位置上有一小数点)。M 可以放一个 0～999.99 之间的任何正数。如执行：

M				
2	1	5	6	3
		Λ		

```
MOVE 215.63 TO M
```

内存中情况如右图。图中 Λ 表示隐含的小数点的位置，传送时按小数点对齐，即小数点与 Λ 位置对齐，然后向左右两边扩展。如传送的数值位数多，则截断，如：

```
MOVE  1215.637  TO  M
```

仍按小数点对齐，向左右二边扩展，在 M 中只能放入 215.63，而将高位 1 和低位 7 舍去。

举例：

描　述		数　值	内　存
02 A PIC 99	V99	87.5	8750 (Λ)
02 T PIC 999	V99	498.5	49850 (Λ)
02 W PIC 9(3)	V9(2)	781	78100 (Λ)
03 N PIC 9(4)		1245.6	1245
03 H PIC	V999	1.234	234 (Λ)

说明：(1) V 在描述符的最后，则等于无小数点。如：

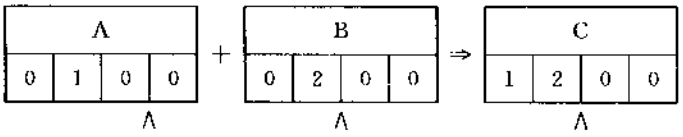
03 M1 PIC 999V.

和 03 M1 PIC 999. ＞等价，

(2) 一个数据的描述符中只能出现一个 V。

如 03 N PIC 99V99V. 是错误的。

(3) 运算时，按隐含的小数点位置对准进行运算。如 A 和 B 在内存中分别为：



即 A 的描述符为 9(3)V9，B 的描述符为 9(2)V9(2)，要进行 COMPUTE C＝A＋B 的运算。若 C 的描述符为 9(2)V9(2)则相加后 C 的内容为1200(Λ)，而不是将 A 与 B 的字节中的内容从左至右相应地相加得 0300。也就是说，V 表示的小数点在内存中是隐含的(不占内存位置)，但是有效的。

(4) 显示时，只将内存中各字节的内容显示出来，由于 V 不占内存，因此如果将上面的 A，B，C 显示出来(DISPLAY A，B，C)，则小数点不会显示出来。显示结果分别是 0100，0200，1200，而不是 010.0，02.00，12.00。即显示时是按内存中信息转换成的字符显示的，用 V 表示的小数点是有效的，但只在传送和运算时起作用，不能被显示和打印出来。如果想打印出小数点“.”，应该用编辑型数据(见本章 § 4.3 节中 4.3.4 段)。

(三) P 描述符

当数值很大，后面若干位为 0，如 1000000000(十亿)，需要用 PIC 9999999999 来描述，占十个字节。为了节省内存，对这种低位上有若干个零的数，可以用“P”描述符。如：

01 A PIC 9PPPPPPPPP

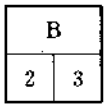
在内存中只占一个字节，内放数字 9，九个 P 表示在数字 9 的后面有九个零，即表示 1×10^9。P 是隐含的，不占内存单元。在运算时，会自动按 1×10^9 进行运算。也可写成：

01 A PIC 9P(9).

对很小的数，如 0.000023，即到小数点后若干位后才有非零数字的数，可以这样描述：

01 B PIC PPPP99.

表示在两个数字前，有四个零，还有一个小数点在最前面的零的前面，如果在内存中 B 为：



由于在 PIC 子句中用 PPPP99 描述，它表示 0.000023。

注意：(1) P 必须出现在全部"9"之前或全部"9"之后，如 99P99 的写法是不正确的，即"P"只能用来表示"前零"或"后零"。在前零前有一小数点。

(2) 用"P"，则表示小数点的位置已隐含地决定了。如：

99PP
99PPV > 等价　　PP99
　　VPP99 > 等价。

V 的位置不应与 P 隐含规定的位置相矛盾。如 PPV99 是错误的，不能用它存放 00.43 这样的数。

(3) 如在"9"后出 n 个 P，表示在内存中的数应乘以 10 的 n 次方。如数据项 A 的描述为：PIC 9(4)P(6)，内存中情况为：

A			
1	2	3	4

它表示 A 的值为 1234×10^{6}。

如果在"9"之前有 n 个 P，则应乘以 10^{-m}，其中 m=n+k。k 为"9"的个数，例如：

02 C PIC P(4)9(4).

在内在中情况为：

C			
1	2	3	4

它表示 C 的值为 1234×10^{-8}。此时 n=4，k=4，m=n+k=8。

举例：

描　述	内存中数字	等价的算术量
03 AMOUNT PIC 9(4)PPPP	1802	18020000
03 WAT PIC 9(2)P(3)	015	15000
03 MAT PIC P(3)9(2)	11	0.00011
03 MAN PIC VPP99 (或 PIC PP99)	87	0.0087

(4) 显示或打印时 P 是不出现的，例如显示上表中的 AMOUNT 时，不是显示出 18020000，而是 1802，即只显示出内存中实际存放的数字而不显示隐含的 P(即零)。

(5) P 描述符在商业上不常用，主要用于科技计算中(但科技计算并非 COBOL 的主要对象)。

(四) S 描述符

如果想在数据项中放入一个带符号的数，可以用"S"描述符。如：

02 D PIC S99.

可以将一个正的或负的数存放在 D 中。

如：　MOVE −12 TO D.

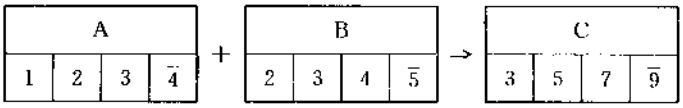
D 的描述已经指定如上，D 在内存中仍只占两个字节，即 S 不计入数据项的长度中。根据数据项的不同描述决定符号在内存中的实际位置。对于用 PIC S99 来描述的数据项，系统在内存中 D 数据项的最后一个字节中，放入一个标记，表示此数为负。

举　例	表示的数值	内存中情况
02 B PIC S9(4)V9(2)	126.89	012689 (∧)
02 C PIC S9(4)V9(2)	−1112.34	11123$\bar{4}$ (∧)
02 D PIC S9(4)V99	−0.25	00002$\bar{5}$ (∧)
02 E PIC S9(4)V99	−727.18	07271$\bar{8}$ (∧)

上表中∧表示隐含的小数点的位置。$\bar{4}$表示对于C这种描述的数据项(外部十进制形式,见第七章§7.1),在4所在的字节中(4为数据项C中最后一个字节),放入一个负号的标记。表示整个数值是负的,即表示−1112.34。

注意:(1) S必须是最左边一个描述符,如9999V99S是不正确的写法。

(2) 在内存中,S不占位数,在运算时,S起作用,如A,B,C均用S9(4)来描述,A的值为−1 2 3 4,B的值为−2 3 4 5,运算后C的值为−3 5 7 9。



但输出时,符号不会输出来,它按内存中实际存放的信息输出。如打印A,打印出前三个字符为1 2 3,打印出的第四个字符,既非4,又非"−",而是一个其它的字符,因为最后一个字节中放的已不是原来的4了,而加入了"负"号的信息。在学习本章时,读者可不必细究,在第七章中将对此作进一步的说明。

4.3.2 字母型数据的描述

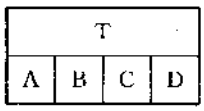
字母型数据项用A描述符,在这种类型的数据项中只允许存放字母或空格。例如:

```
02  T  PIC  AAAA.
```

表示T中可放入四个字母。如果在过程部中有:

```
MOVE 'ABCD'  TO  T
```

则内存中T数据项内容为:



注意不能放入字母和空格以外的字符,如欲放入"A BC"或"A·CB","A123"等都是不对的。一个字母在内存中占一个字节。

【例4.2】

```
IDENTIFICATION      DIVISION.          (环境部)
PROGRAM-ID.         EXAM42.
ENVIRONMENT         DIVISION.
INPUT-OUTPUT    SECTION.
FILE  CONTROL.
    SELECT    PRINT-FILE  ASSIGN  TO  PRINTER.    (打印机名)
DATA  DIVISION.  (数据部)
FILE   SECTION.  (文件节)
```

```
FD    PRINT-FILE  LABEL  RECORD  IS OMITTED.
01  T.
  02  FILLER  PIC  X.
  02  T1  PIC  A(5).
  02  T2  PIC  A(3).
  02  T3  PIC  A(2).
  02  T4  PIC  A(5).
PROCEDURE  DIVISION.      (过程部)
  OPEN  OUTPUT  ABC.      (打开输出文件 ABC)
  MOVE 'THIS  ' TO  T1.
  MOVE 'IS  ' TO  T2.
  MOVE 'A  ' TO  T3.
  MOVE 'BOOK  ' TO  T4.
  WRITE  T  AFTER 1.
  CLOSE  ABC.
  STOP  RUN.
```

在执行完 MOVE 语句后记录 T 中的内容如下：

T														
FILLER	T1					T2			T3		T4			
	T	H	I	S		I	S		A		B	O	O	K	

执行 WRITE T 语句时，在打印机上打印出：THIS IS A BOOK 。

4.3.3 字符型数据的描述

（一）字符型数据的规定

往往需要在数据项中放入一些既非数字又非字母的字符，比如，想打印出“1/5/1992”或“J·ALOT”，需要把“/”或“.”放入数据项中，打印时按原样印出。这种由任意的 COBOL 字符组成的数据，称字符型数据，它是非数值类型的，不能用于运算。

可以用 X 描述符来描述字符型数据。如：

```
03  A  PIC  X(3).
```

表示可在数据项 A 中放入三个任意的 COBOL 字符，如下图(a)示。

如果 MOVE 'NO' TO A，即送入的字符少于 A 的字节数，则数据项中最右边一个位置补以空格。见下图(b)。

A		
Y	E	S

(a)

A		
N	O	

(b)

描　述	送入的数据	内存中情况
02 R1 PIC X(4)	BOOK	BOOK
02 R2 PIC X(8)	SIN(X)	SIN(X)␣␣
02 R3 PIC X(7)	COBOL-74	COBOL-7
02 R4 PIC X(12)	DATA-NAME	DATA-NAME ␣␣␣

说明：(1) 字符型数据可以用 X 描述符来描述，也可以用 9 和 A 描述符来描述。如欲将 COBOL-74 八个字符送到 R3 中，既可以用：

```
02  R3  PIC  X(8).
```

描述，也可以用：

```
02  R3  PIC  A(5)X9(2).
```

来描述，在其中都可以容纳“COBOL-74”字符。

又如想将 SIN(X)放入 R2 中，则 R2 的描述可以用：

```
02  R2  PIC  X(6).
```

也可以用：

```
02  R2  PIC  A(3)XAX.
```

因前三个和第五个字符为字母，第四和第六个为非字母和非数字的字符，只能用 X 描述符。

可以看出：用 A(5)X9(2)来描述的数据项 R3，不会是数值型的，也不是字母型的(因字母型的只能用 A 描述符)，它的值是 COBOL-74，它属于字符型的。当混合使用描述符 9，A 或 X 时，数据必定为字符型的，譬如用 99X99，A99，AXA 等描述的都是字符型的数据项。

数值型数据的描述只能用 9，V，S，P 描述符。字符型数据的描述中可以用 9，A，X。

(2) 字母型数据既可用 A 描述，也可用 X。

如：

```
02  N  PIC  A(4).
```

也可以用下面的描述来代替：

```
02  N  PIC  X(4).
```

有人可能会认为：既然字符型包括了字母，何必再用字母型数据项呢？有时当某一数据项中的数据全为字母时，为了保证数据的正确，用 A 描述符。当向它送入非字母时，系统就会报告出错信息，容易出现错误。

(3) 字符型数据中可以放数字。如：

```
02  K  PIC  X(3).
```

可以使用 MOVE '123' TO K 语句，将三个字符 1，2，3 送入 K 中三个字节。而：

```
02  M  PIC  9(3).
```

M 为一数值型数据，如果用：

```
MOVE  123  TO  M.
```

M 中也放了 1 2 3。

K		
1	2	3

M		
1	2	3

那末 K 中 1 2 3 和 M 中的 1 2 3 有什么不同？K 是字符型的，在它三个字节中放入字符 1，2，3，它只是三个字符，而不是代表“一百二十三”这一数值。也就是说尽管在数据项中全部放了数字，但由于它是用 X(而不是用 9)描述的，因此它只能存“字符”，而不能存“数值”。在 K 中放入“1 2 3”，和放入“ABC”，性质是一样的，它们仅仅是三个孤立的字符。

用 MOVE 向一字符型数据项送入字符时，应该用引号括起来，如：

```
MOVE 'ABC' TO  K.
MOVE '123' TO  K.
```

而用 MOVE 向一数值型数据送入一个数值时，不应使用引号。如

MOVE 123 TO M.

前者('ABC','1 2 3')是非数值常量,而后者(1 2 3)是数值常量。如果写成:

MOVE 123 TO K.

或 MOVE '123' TO M.

都是错误的。读者应严格区别数值型与非数值型数据的性质和用法。只有数值型数据才能用于计算。非数值型数据(如字母型、字符型)不能用于计算,往往只是为了打印的需要,才使用这种数据项。

1 2 3+4 5 6 是合法的,而'1 2 3'+'4 5 6'是错误的。同样,如果 K 和 L 都是字符型数据,内放的字符分别为 1 2 3 和 4 5 6,则:

ADD K TO L

也是错误的。

下面举例说明以上介绍的几种类型数据的使用。

【例 4.3】 在数据部中描述职工工资记录。设职工工资数据存放在一个磁盘文件中,假定该文件的内部文件名为 INPUT-FILE,记录名为 GZQD-R(以汉语拼音表示的"工资清单","R"是记录的缩写)。每个职工的工资清单为一个记录,它包括职工编号、姓名、基本工资、附加工资、副食补助、洗理费、托儿费、房租、互助金、病事假扣除等项。

在数据部中可以描述如下:

```
DATA  DIVISION.        (数据部)
FILE  SECTION.         (文件节)
FD  INPUT-FILE
        LABEL  RECORD  IS  STANDARD.
01  GZQD R. (工资清单)
    02  BH (编号)              PIC  9(6).
    02  XM (姓名)              PIC  X(10).
    02  JBGZ (基本工资)        PIC  9(3)V99.
    02  FJGZ (附加工资)        PIC  9V99.
    02  FSBZ (副食补助)        PIC  9V9
    02  XLF (洗理费)           PIC  9V99.
    02  TEF (托儿费)           PIC  99V99.
    02  FZ (房租)              PIC  99V99.
    02  HZJ (互助金)           PIC  99V99.
    02  BSJKC (病事假扣除)     PIC  99V99.
```

记录的结构和在内存中存放的顺序如下表所示。

记录名	GZQD-R 工资清单记录									
初等项	BH	XM	JBGZ	FJGZ	FS BZ	XLF	TEF	FZ	HZJ	BSJKC
	编号	姓名	基本工资	附加工资	副食补助	洗理费	托儿费	房租	互助金	病事假扣除
描述	9(6)	X(10)	9(3)V99	9V99	9V9	9V99	99V99	99V99	99V99	99V99
相对地址1	6 7	16 17	21 22	24 25	26 27	29 30	33 34	37 38	41 42	45

数据从磁盘文件中输入，某一个记录的数据形式如下：

000001WANG FUNG12856 550 50 150 1027 0543 0500 0435

6 10 5 3 2 3 4 4 4 4

在输入时，记录中第1至第6列的数据"000001"送入数据项BH中。设相对地址从1开始，BH占相对地址1～6的一段内存区，6个字节。记录中第7～16列中的数据"WANG FUNG"送入XM中，相对地址为7～16。记录中第17～21列数据输入到JBGZ中去。注意输入数据中不应包括小数点(不应输入128.56，而应输入12856)，输入时，依次将12856输入到JBGZ中第一个到第五个字节中，在第三个字节后有一隐含的小数点，即已输入的值为128.56。后面的数据依此类推。

4.3.4 编辑型描述符

如前所述，如果有：

```
02  H  PIC  999PPP.
```

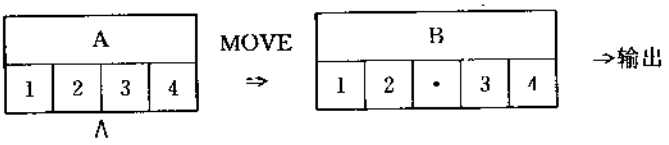
若在内存中H的三个字节内存放了135，它表示数值135000，但输出来的是135。又如：

```
02  Q  PIC  9(6).
```

如果Q的值为1，数据项中六个字节分别放"000001"，输出来是000001。虽然没错，但前面5个零不好看，最好能去掉，而只输出1。

此外，在商业、管理工作中，有时需要用到一些专门的符号，如在数值前加"$"，譬如$120，$5600等。或者希望在打印68000000这类比较大的数时，每三位间加一分位号，如68,000,000。这样使打印结果符合人们的习惯，看起来清楚些。人们希望得到的打印结果不需再加工，就能作为正式文件使用(如：作为付款通知书、收据、统计表等)。因而对打印格式有较高的要求。

为了解决这一问题，就需要另设一个数据项，在其中安排输出时所需的内容(如加上"$"，+，－等符号)。COBOL专门有一种形式的数据项，它不是作计算用的，而是只作为输出数值数据时增加或改变某些所需的符号，起"编辑"的作用，如将内存中12∧34，先变为12.34，然后再输出12.34。在输出前，先把想输出的数值数据送入这个编辑数值数据项中，然后



再将这个编辑数值数据打印出来。上图中A为数值型数据项，B为编辑数值型数据项，它对数值型数据进行"编辑"，加上一个明显的(而不是隐含的)小数点。输出B(而不是输出A)时，就能得到12.34。

再强调一下：编辑型数据项(例如上面的B)仅仅是为了输出的需要，没有其它作用，它不能用来运算。如ADD A TO B是错误的。如果不输出数据，或对输出格式要求不高(如输出都是整数，用不到加小数点，也不必加$等)，可以用不到编辑型数据项。

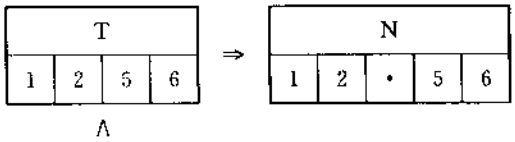
下面分别介绍编辑数据项中用到的编辑描述符。

(一) 插入小数点"."，用"."描述符

使数值型数据中隐含的小数点在输出时能在相应的位置上打印出“.”来。如：

```
77  T   PIC   99V99.   (T 为数值型数据项)
  ⋮
77  N   PIC   99.99.   (N 为编辑型数据项)
```

如 T 的内容为 12 56，执行 MOVE T TO N，则 N 的内容为 12.56。T 在内存中占 4 个字节，而 N 占 5 个字节。注意，并不是在数据项 T 中加小数点，而是另增设一个数据项 N，N 是

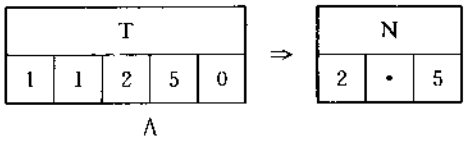


5 个字节(因多设了一个小数点位置)。将 T 的内容送 N 时，就按小数点对齐，将隐含的小数点位置对在 N 中的小数点处，然后向左右两边扩展，最后得到 12.56。这时如果用 DISPLAY N 显示 N，则得到 12.56 五个字符。

如果 T 和 N 分别描述为：

```
77  T   PIC   999V99.   (数值型数据)
  ⋮
77  N   PIC   9.9.   (编辑型数据)
```

T 的值为 112.56，N 只有三个字节，而且其中包括一个小数点。则按小数点对齐后，多余位的数舍去。如下图。



(二) 插入逗号“,”作分位号，用“,”描述符

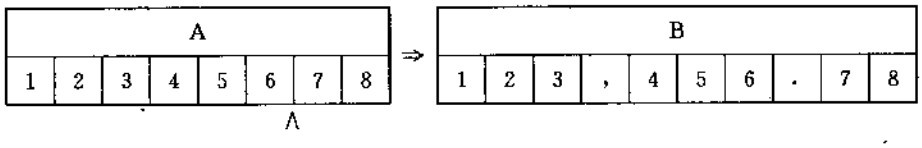
如有：

```
77  A   PIC   9(6)V99.
```

A 的值为 123456.78，在内存中为 12345678，希望输出结果为 123,456.78，即在百位数字左面加一逗号，在个位数后加一小数点，可设编辑型数据项 B：

```
77  B   PIC   999,999.99.     (或 PIC 9(3),9(3).9(2))
```

执行 MOVE A TO B，则 B 的内容为 123,456.78。



用 DISPLAY 输出 B，显示出 123,456.78。

(三) 插入零，用“0”描述符

数值型数据的描述中有 P，希望在输出时能出现相应的“0”。如：

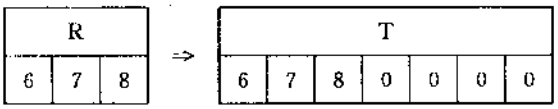
```
77  R   PIC   999PPPP.
```

它在内存中放 678，即值为 6780000。希望输出 6780000 而不是 678。需要在 678 后面人为地

插入四个零。

77 T PIC 9990000.（或9(3)0000,或9(3)0(4)）

执行 MOVE R TO T.

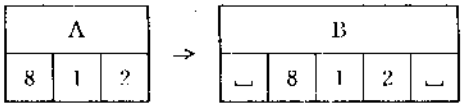


显示 T,得 6780000。R 占 3 个字节,T 占 7 个字节,每一个"0"占一字节。

（四）插入空格,用"B"描述符

77 A PIC 9(3). （数值数据项）
77 B PIC B9(3)B. （编辑数据项）

如 A 值为 812,在执行 MOVE A TO B 语句后,B 中内容为␣812␣。



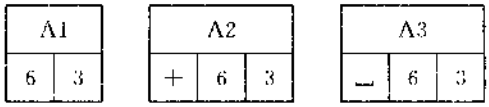
若输出 B,打印结果为␣812␣。

（五）插入正负号,用"+"或"−"描述符

往往需要在输出的数值前(或后)加正号或负号。如+123.6,−56.37 等。

77 A1 PIC S99. （数值数据项）
77 A2 PIC +99. （编辑数据项）
77 A3 PIC -99.

如 A1 值为+63,则执行 MOVE A1 TO A2,A3 后,A2 内容为+63,A3 内容为␣63。



规则:当用描述符"+"时,不论数值为正或负,一律加符号。当用"−"时,当数值为正时,数值前空一格,数值为负时,加一负号。

描述符	数值为正	数值为负	说　明
+	加"+"号	加"−"号	一律加符号
−	数前空一格	加"−"号	只对负值加负号

符号也可以加在数值的后面　如用:02 A2 PIC 99+,则输出为 63+。

举例:如执行 MOVE B1 TO B2,结果如下:

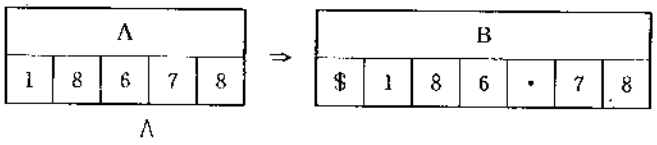
打印结果 / B2的描述 / B1 值	+9999	-9999	9999+	9999−
+1268	+1268	␣1268	1268+	1268␣
−1268	−1268	-1268	1268−	1268−
0(按+0处理)	+0000	␣0000	0000+	0000␣

（六）插入“$”

（1）把$加在数字前

```
77  A   PIC   9(3)V99.      (数值数据项)
77  B   PIC   $999.99.      (编辑数据项)
```

如A值为186.78，内存中为18678，执行MOVE A TO B后，B的内容如下图所示。



A

输出B，则打印结果为$186.78。

（2）在数字前加正负号和$

“+”或“-”为PIC子句中的第一个描述字符，“$”为第二个描述字符。

```
77  A   PIC   S99V9.      (数值数据项)
77  B   PIC   +$99.9.     (编辑数据项)
77  C   PIC   -$99.9.
```

如A的内容为125，即值为12.5，则执行MOVE A TO B，C后，B的内容为+$12.5，C的内容为␣$12.5。如A值为-12.5，则B内容为-$12.5，C内容为-$12.5。B和C各占6个字节。A占3个字节。

（七）浮动插入正负号和“$”

如果有：

```
77  A1  PIC  S9(3)V 99.     (数值数据项)
77  A2  PIC  +9(3).99.      (编辑数据项)
77  A3  PIC  $9(3).99.      (编辑数据项)
```

如果A1值为2.12，执行MOVE A1 TO A2，A3后，输出A2和A3的结果为：+002.12和$002.12，看起来不理想，希望取消高位零而输出结果：+2.12和$2.12。也就是使$或+、-号紧挨着数字之前。如果A1的值为12.12，则希望输出结果为：+12.12和$12.12。也就是说$或+、-号的位置不是固定的，而是浮动的，希望浮动的范围足够大，使得$或+、-号能紧挨在数字之前。即$（或+、-号）出现在最高位非零数字之前。

对上面这个问题，我们改用下面的描述：

```
77  A2  PIC  +++++.  99.
77  A3  PIC  $$$$.  99.
```

出现了两个以上的“+”（或“-”）号和“$”号，它的作用是：‘$’或‘+’的位置是浮动的。浮动的范围是在PIC子句中指定的四个“$”（或四个“+”）之间。即“$”或“+”的位置随着数值的最高位置的变化而变化。但它也有一定范围，它可以是第一个位置到第四个位置之一，好比“水涨船高”。也就是说，在浮动的“$”（或“+”、“-”）位置上可以送入数字，然后取消高位零，并在数字前加一个“$”（或“+”、“-”）符号。当A1的值为49.12，在执行MOVE A1 TO A2，A3后，A2中正号‘+’的位置在左面第二个字符，即␣+49.12。当A1值为149.12时，A3的内容为$149.12。

1. 浮动插入“$”

执行 MOVE A TO B 的情况见下表：

A 的描述	A 数值	B 的描述	B 内容
PIC 9(3)V99	21.89	PIC $$99.99	␣$21.89
PIC 9(3)V99	12.62	PIC $(3)9.99	␣$12.62
PIC 9(3)V99	0.15	PIC $$9.99	␣␣$0.15
PIC 9(3)V99	0.15	PIC $$$.$$	␣␣␣$.15
PIC 9(3)V99	0.0	PIC $$$.$$	␣␣␣␣␣␣␣
PIC 9(3)V99	0.05	PIC $$$.$$	␣␣␣$.05

注意：(1) 数据项 B 的描述可以全部用“$”(小数点除外)，即对应于所有的数字字符。若数值为 0，则得到全部空格，连描述字符“.”的位置上也是空格。

(2) 如数值是一小数，则 $ 只能向右浮动到小数点位置。见上表中最后一例。即 $ 不能越过小数点而成，$5 或 $5，这就使原来 0.05 变成 $5 了，不符合人们要求。

2. 浮动插入“+”或“−”号

与 $ 类似，若执行 MOVE A TO B，则：

A 值	B 的描述	B 内容	
20.3	PIC +999.9	+020.3	(固定的+号)
−20.3	PIC 999.9+	020.3−	
530.83	PIC −999.99	␣530.83	
−530.83	PIC 999.99−	530.83−	
−1.3	PIC +++9.9	␣␣−1.3	(浮动插入符号)
−1.03	PIC ++++.++	−1.03	
0.0	PIC ++++.++	␣␣␣␣␣␣	(全空，连小数点位置也变成空格)
−21.86	PIC −−99.99	␣−21.86	
10.44	PIC −−99.99	␣␣10.44	

注意：(1) 在编辑型数据的描述中，指定浮动插入的字符个数应足够，以免数据被截断。如：

```
77  A  PIC  9(4)V99.      (数值数据项)
77  B  PIC  $(4).99.      (编辑数据项)
```

若 A 值为 8764.31，执行 MOVE A TO B 时，B 的内容为 $764.31。第一个数字“8”被截掉了。原因是 $(4).99 不够用，应改为：

```
77  B  PIC  $(5).99.
```

(2) 浮动字符“$”或“+”、“−”前不能再出现其它符号，即浮动字符应为第一个描述字符。

```
77  A1  PIC  +$$.99.
```

是错误的。如想加正、负号，可以放在数字的后面，如

```
77  A1  PIC  $$.99+.
```

或 77 A1 PIC $$.99−.

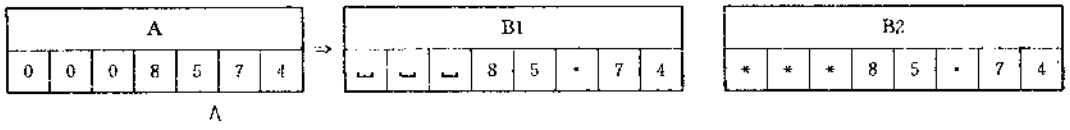
是合法的，因为这样的正负号不会影响$的浮动。

(八) 取消高位零，用"Z"和"*"描述符

用$，+，−浮动，可以取消高位零，但在数值前出现$或+，−号。如果只要求取消高位零，不需加$"+，−"，则用"Z"或"*"描述符。在高位零的位置上代以空格式"*"。如有：

```
77  A    PIC  9(5)V99.
77  B1   PIC  Z(5).99.
77  B2   PIC  *(5).99.
```

若A值为85.74(内存中为0008574)，则执行MOVE A TO B1，B2后，B1和B2内容分别为：␣␣␣85.74和***85.74。



连续的"*"号用作票据的"保护符号"，以防涂改数值。

注意：(1) 不能同时用Z浮动和+，−，$浮动。

如用Z Z$ $99.99或$ $Z Z99.99都是不对的。

(2) 单个的+，−，$可以与Z或*浮动连用。

```
77  A1  PIC  S9(3)V99.
77  A2  PIC  +Z(3).99.
77  A3  PIC  −*(3).99.
77  A4  PIC  $Z(3).99.
77  A5  PIC  $*(3).99.
```

当A1值为+2.58时，执行MOVE A1 TO A2，A3，A4，A5后，A2，A3，A4，A5的内容分别为：+␣␣2.58，␣**2.58，$␣␣2.58，$**2.58，可以将它们输出。如果A1值为0(按+0处理)，则A2，A3，A4，A5的内容分别为+␣␣␣.00，␣***.00，$␣␣␣.00，$***.00。

(3) 如果使"Z"或"*"对应于所有的数字字符，如Z(4).Z(3)或*(5).*(2)，当数值为0时，则所有数字位全部由空格或*代替。小数点位置上，由空格代替(用Z描述符时)，或保留"."(用*描述符时)。如：

```
77  A    PIC  9(4)V99.     (数值数据项)
77  A1   PIC  Z(4).ZZ.     (编辑数据项)
77  A2   PIC  *(4).**.     (编辑数据项)
```

执行MOVE A TO A1，A2后，当：

A值为0.0时，A1内容为␣␣␣␣␣␣␣，A2内容为****.**。

A值为0.01时，A1内容为␣␣␣␣.01，A2内容为****.01。

(4) "Z"，"*"可与","一起使用，但当插入的","前边是被取消的无用零时，该","位置也被␣或"*"代替。

如：执行MOVE A TO B，则

A 值	B 的描述	B 内容
58	PIC Z99	␣58
85	PIC ZZZ.99	␣␣85.00
0.012	PIC ZZZ.ZZZ	␣␣␣.012
0.0	PIC ZZZ.ZZ	␣␣␣␣␣␣␣
0.0	PIC $Z(3).ZZ	$␣␣␣␣␣␣
0.0	PIC ***.**	***.**
2.6	PIC **.*	*2.6
652.32	PIC *.***.**	**652.32
1812.43	PIC Z,ZZZ,ZZZ.ZZ	␣␣␣␣1,812.43

（九）插入“DB”和“CR”字符

在银行业务中，有时用到“DB”(debit，借方)和“CR”(credit，贷方)。

DB 和 CR 只能用作固定插入，而且只作为最后一个描述符号。当数值为负时，在编辑型数据项中最后两个字节中置 DB 或 CR，数值为正时，此两字节留空格。

例如，执行：MOVE A TO B，则

A 值	B 的描述	B 内容
512.12	$9(3).99DB	$512.12␣␣
-512.12	$9(3).99DB	$512.12DB
138.57	$999.99CR	$138.57␣␣
-138.57	$999.99CR	$138.57CR

编辑字符除了可用于数值型数据的编辑外，还可以用于字符型数据的编辑。但是，显然以上介绍的有些编辑字符是不能用于编辑字符型数据的(如小数点，+，-，$，Z，CR，DB等)。可用于字符型数据的编辑字符为 B 和 0 字符，即用来插入空格和插入零字符。

例：执行 MOVE A TO B 后，B 的内容如下：

数据项 B 的描述	数据项 A 内容	B 的内容
AAABAAAA	NEWYEAR	NEW␣YEAR
X(4)BX(2)BX(2)	19911015	1991␣10␣15
X(5)B(3)	CHINA	CHINA␣␣␣
00X(6)	PEOPLE	00PEOPLE
00X(4)	1357	001357
BX0X(3)B	X123	␣X0123␣
ABABABA	WORK	W␣O␣R␣K

COBOL 中用编辑字符来描述数据，主要是为打印输出服务的。通过对数值数据进行编

辑，使输出结果符合会计、银行、统计工作等行业的习惯形式，使打印报表美观、灵活、易懂，COBOL 在这方面具有丰富的表达能力，这是比其它语言优越的地方。为了能表达多种多样的报表格式，数据编辑的功能就自然比较复杂一些。对初学者来说，开始不必要一下子全部掌握，更不要死背硬记。以后用到时临时查一下即可。每一行业都有自己的常用打印报表要求，只要掌握所需的部分即可。

4.3.5 PIC 子句小结

（一）PIC 子句用来说明数据的类型和长度。PIC 子句的一般格式为：

$$\left\{\begin{matrix}\underline{\text{PICTURE}}\\ \underline{\text{PIC}}\end{matrix}\right\}\ \text{IS}\ \ \text{描述字符串}$$

PIC 子句只能用来描述初等数据项。

（二）每一种类型数据可以使用的描述字符如下：

数据类型	在 PIC 子句中允许使用的描述字符
数值数据项	9 V S P
字母数据项	A
字符数据项	9 A X
编辑数值数据项	9PV.,BZ+ − $ * 0 CR DB
编辑字符数据项	A X 9 B 0

注：数值型数据项还可以用“外部浮点形式”描述，详见第七章（下册）。

（三）描述字符的含义

描述字符	含义
9	表示一个数字位置
A	表示一个字母位置
X	表示一个字符位置
V	表示隐含的小数点位置
S	表示数值数据带符号
P	表示十进比例换算，即指明落在数据域外的十进小数点位置
$	插入货币号位置
.	插入小数点位置
,	插入逗号的位置
+	一律加符号
−	对负数加负号，对正数前留一空格
Z	取消高位零，代以空格
*	取消高位零，代以 *
B	插入空格的位置
0	插入零的位置
DB(借方)	数据为负时，在数据后面出现 DB，数据为正时，数据后空两格
CR(贷方)	数据为负时，数据后出现 CR，数据为正时，数据后空两格

（四）程序举例

【例 4.4】 从磁盘数据文件中读入职工的工资记录，计算出实发工资，输出到磁盘上，建立工资文件。需要时可通过打印机打印该工资文件中的数据。

这个程序的数据部已在例 4.3 中列出。下面列出完整的程序。在程序输出结果时使用了编辑型数据项。

```
IDENTIFICATION    DIVISION.                          (标识部)
PROGRAM-ID.  EXAM4 4.
ENVIRONMENT    DIVISION.                             (环境部)
INPUT-OUTPUT    SECTION.                             (输入输出节)
FILE CONTROL.                                        (文件控制段)
    SELECT  INPUT FILE  ASSIGN TO IN-FILE.
    SELECT  PRINT FILE  ASSIGN TO PRI-FILE.
DATA    DIVISION.                                    (数据部)
FILE  SECTION.                                       (文件节)
FD  INPUT FILE  LABEL  RECORD IS STANDARD. (文件名为 INPUT-FILE)
01  GZQD R.                                         (记录名)
    02  BH     PIC 9(6).                            (编号)
    02  XM     PIC X(10).                           (姓名)
    02  JBGZ   PIC 9(3)V99.                         (基本工资)
    02  FJGZ   PIC 9V99.                            (附加工资)
    02  FSBZ   PIC 9V9.                             (副食补助)
    02  XLF    PIC 9V99.                            (洗理费)
    02  TEF    PIC 99V99.                           (托儿费)
    02  FZ     PIC 99V99.                           (房租)
    02  HZJ    PIC 99V99.                           (互助金)
    02  BSJKC  PIC 99V99.                           (病事假扣除)
FD  PRINT FILE  LABEL  RECORD IS STANDARD  (文件名为 PRINT FILE)
01  GZQD P.                                         (记录名)
    02  FILLER PIC X.
    02  BH-P       PIC 9(6).                        (编号)
    02  FILLER PIC X.
    02  XM P       PIC X(10).                       (姓名)
    02  FILLER PIC XX.
    02  JBGZ-P     PIC 9(3).99.                     (基本工资)
    02  FILLER PIC XX.
    02  FJGZ P     PIC 9.99.                        (附加工资)
    02  FILLER PIC XX.
    02  FSBZ-P     PIC 9.9.                         (副食补助)
    02  FILLER PIC XX.
    02  XLF P      PIC 9.99                         (洗理费)
    02  FILLER PIC XX.
    02  TEF-P      PIC 99.99.                       (托儿费)
    02  FILLER PIC XX
    02  FZ-P       PIC 99.99.                       (房租)
    02  FILLER PIC XX.
    02  HZJ-P      PIC 99.99.                       (互助金)
    02  FILLER PIC XX.
    02  BSJKC-P    PIC 99.99.                       (病事假扣除)
    02  FILLER PIC XX.
    02  SFGZ-P     PIC 9(4).99.                     (实发工资)
PROCEDURE    DIVISION.                              (过程部)
```

```
K.  OPEN  INPUT    INPUT-FILE
          OUTPUT   PRINT FILE.
D.  MOVE  SPACE  TO  GZQD-P.
    READ  INPUT-FILE
          AT END CLOSE INPUT FILE,PRINT FILE
                 STOP  RUN.
S.  MOVE  BH     TO BH P.
    MOVE  XM     TO XM-P.
    MOVE  JBGZ   TO JBGZ-P.
    MOVE  FJGZ   TO FJGZ-P.
    MOVE  FSBZ   TO FSBZ P.
    MOVE  XLF    TO XLF-P.
    MOVE  TEF    TO TEF P.
    MOVE  FZ     TO FZ-P.
    MOVE  HZJ    TO HZJ-P.
    MOVE  BSJKC  TO BSJKC P.
SU. COMPUTE SFGZ P = JBGZ + FJGZ + FSBZ + XLF - TEF - FZ - HZJ - BSJK
X.  WRITE  GZQD P  AFTER 2.
    GO TO D.
```

输入文件 IN-FILE 中记录的数据形式如下：

```
000001LI          LI1002278880605150005000200l000      (第一个记录)
000002WANG   HONG2002280075500557708670366010 9      (第二个记录)
000003MA        MIN150889507568810501550020000000      (第三个记录)
                          ⋮
```

程序的流程是：先执行 K 段(用'K'表示'开')，打开文件。再执行 D 段(用'D'表示'读')，将输出文件的记录区内容"冲空"，然后读入一个输入文件的记录。再执行 S 段(以'S'表示'送')。把输入记录中的数据项传送到输出文件的记录区 GZQD-P 中去。见图 4.1。

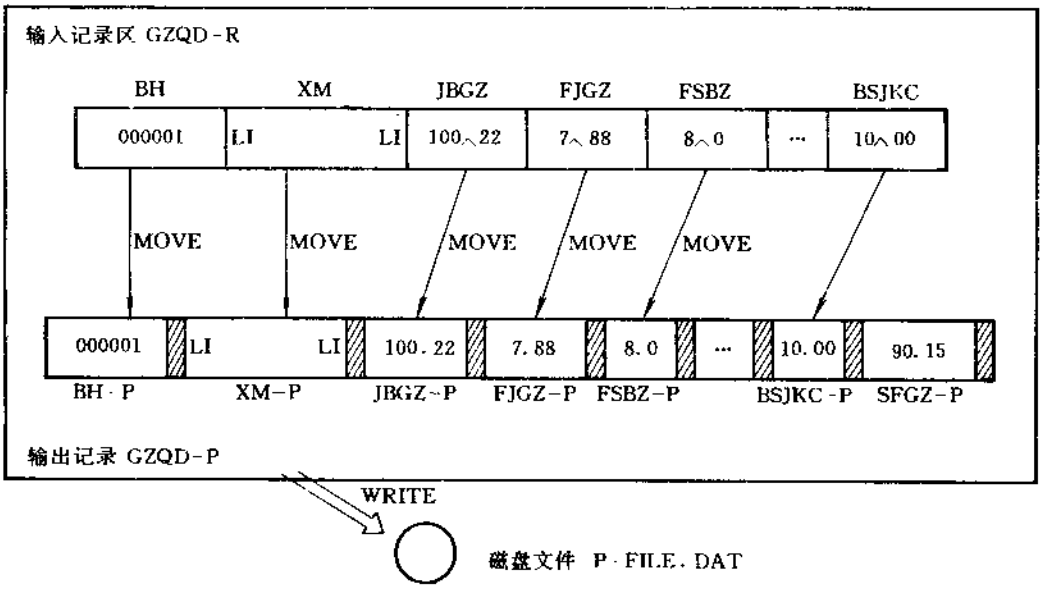


图 4.1

输出记录区和输入记录区中的各项是相应的。在输出记录区中，也有编号、姓名、基本工资等项，用 BH-P，XM-P，JBGZ-P……表示，BH-P 中的 P 表示 PRINT，即打印，这样起名，既可看出是代表"编号"，又表示是输出记录中的一个数据项，以与输入记录区中的"编号" BH 相区别。输入记录区中的 JBGZ 是数值型数据项，而输出记录区中的 JBGZ-P 是编辑数值型数据项，在 JBGZ 向 JBGZ-P 传送时，将数值型数据转换成编辑数值型数据（将隐含小数点转换为明显的小数点）。在输出记录中各数据之间插入一些空格（FILLER 内放空格），以便打印时美观、清楚。

接着执行 SU 段（SU 表示'算'）。计算实发工资，公式是：实发工资＝基本工资＋附加工资＋副食补助＋洗理费－托儿费－房租－互助金－病事假扣除。

执行 X 段（以'X'表示写）。PRINT FILE 已在环境部与外部文件 P-FILE.DAT 建立联系。执行 WRITE 语句，将输出记录区的内容输出到 PRI FILE 文件，然后再返回 D 段，将输出记录区冲空，再读入一条记录，接着执行 S 段（传送）、SU 段（计算）、X 段（写），直到遇到文件结束标志，执行 AT END 后面所指出的语句，关闭文件，停止运行。

磁盘文件 PRI-FILE.DAT 中的数据如下。可以通过打印机把这些数据打印出来。

```
000001 LI          LI   100.22   7.88   8.0   6.05    15.00   05.00   02.00   10.00   0090.15
000002 WANG      HONG   200.22   8.00   7.5   5.00    55.77   08.67   03.66   01.09   0151.53
000003 MA          MIN  150.88   9.50   7.5   6.88    10.50   15.50   02.00   00.00   0146.76
```

为了便于理解，程序中在编辑型数据项中只用了小数点编辑符，读者可以将它们作进一步的改进，用浮动的编辑符"Z"来使打印出来的数值中的前导零变成空格。

§4.4 工作单元节（WORKING-STORAGE SECTION，又译作工作存储节）

4.4.1 工作单元节的作用

程序中用的数据项分两部分，一部分是属于输入或输出文件的，如例 4.4 中的所有数据项都是输入或输出的数据，这些数据项都应在数据部的文件节中加以描述。另一部分是非输入或输出的数据。例如运算过程的中间结果，或用作累计数的数据项等，这些数据项则在工作单元节中描述。此外，还可利用工作单元节为某些数据赋以初值（用 VALUE 子句）。

在工作单元节中描述的数据项也有两种形式：一种是孤立的数据项，它们是初等项。一种是组合项。COBOL 规定，孤立的数据项的描述体以层号 77 开头，组合项描述体以 01 到 49 之间的一个数作层号。在次序上常先写 77 层，再写 01～49 层。如：

```
WORKING-STORAGE SECTION.
77  A   PIC   XX.
77  B   PIC   99.
01    T.
    02   T1.
      03   T11   PIC   XX.
      03   T12   PIC   99.
    02   T2        PIC   999.
```

4.4.2 赋初值子句（VALUE 子句）

程序中的数据项往往需要赋以初值（如，用作累计数的数据项的初值应为零，某些数据

项要先“冲空”等)。当然可以用:

```
MOVE  0  TO  A.
MOVE  SPACE  TO  B.
```

来实现。但 COBOL 允许直接对工作单元节中的数据赋以初值。如:

```
77  A  PIC  99  VALUE  IS  0.                    (A 的初值为零)
77  B  PIC  XX  VALUE  IS  SPACE.                (B 的初值为两个空格)
01  T.
  02  T1  PIC  S99V99  VALUE  IS  1.2.           (T1 的初值为 1.2)
  02  T2  PIC  X(9)  VALUE  'ZHANG  LIN'.        (T2 的初值为“ZHANG LIN”)
  02  T3  PIC  AAAX99  VALUE 'CHI 56'.           (T3 的初值为“CHI 56”)
  02  T4  PIC  A9X9A  VALUE 'G1.2T'.             (T4 的初值为 “G1.2T”)
```

注意:T3 和 T4 的描述符是 A,9,X 的混合使用,因此,它不是数值型或字母型的,而是字符型的。

说明:(1) 只有对工作单元节中的数据项可以赋初值。不能对文件节中输入输出文件中的数据项赋初值。

(2) 如果在组合项的描述体中使用 VALUE 子句,初值只能是表意常量或非数值型常量。如:

```
02  A  VALUE  '123456'.
  03  A1  PIC  99.
  03  A2  PIC  99.
  03  A3  PIC  99.
```

是可以的。它分别将字符‘12’、‘34’,‘56’赋给 A1,A2,A3。由于 A1,A2,A3 中放的是数字,且描述为 9 型,因此可以用来进行运算,如 ADD A1 TO A2.

不能将 A 的描述体写成:

```
02  A  VALUE  123456.
```

请记住,对组合项整体而言,一律按字符型数据项处理。

(3) 当用一个带符号的数值作初值时,相应的 PIC 子句中应该有“S”描述符。如:

```
77  N  PIC  S99  VALUE  -21.
```

否则,符号无效。

(4) 赋初值时应注意类型的一致性。如:

```
77  D  PIC  X(4)  VALUE  1.22.
```

是错误的,因 1.22 是数值,应该置入数值型数据项中。而

```
77  D  PIC  X(4)  VALUE  '1.22'.
```

是正确的。

表意常量中只有 ZERO 和 ZEROES 既可作为非数值常量,又可作为数值常量使用。如:

```
77  A  PIC  9(6)  VALUE  ZERO.        (A 值为 0,内存中放 000000)
77  B  PIC  X(6)  VALUE  ZEROES.      (B 中内容为六个零字符 000000)
```

(注意:尽管 A 和 B 中内容都是 000000,但它们的性质是不同的。A 是数值型的,可用于计算。B 是非数值型的。不能用于计算。)

(5) VALUE 子句给出的值应适合 PIC 子句描述的范围,否则会出现截断或产生错误。如:

```
77  A1  PIC  S99  VALUE  12.5.          (A1 的值为+12)
```

```
77  A2  PIC  9(2)    VALUE  125.        (A2 的值为 25)
77  A3  PIC  X(5)    VALUE  'PEOPLE'.   (A3 的值为"PEOPL")
77  A4  PIC  9(3)    VALUE  12300.      (A4 的值为 300)
```

§4.5 区 域 图

"区域图"是用来形象地表示数据在内存中的存储情况的。它能使我们对数据部中的数据描述和过程部中语句的执行有一个形象的概念。

只要程序设计者在数据部定义(描述)一个数据项,程序编译时就在内存的用户区中为这个数据项开辟一个区域来存储它的值。例如有:

```
01  A  PIC  X(80).
```

则在内存区中就为数据项 A 开辟一个 80 字节的域,以便放入 A 的值(80 个字符)。

前面已学过,在数据部内的文件节(File Section)中描述属于输入文件和输出文件的记录,在工作单元节(Working Storage Section)中描述非输入输出的数据项。我们可以画出"内存用户区"的区域图,其中分为三个区域,即输入记录区,输出记录区和工作单元区,见图 4.2。

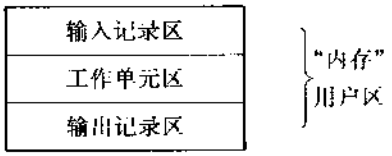


图 4.2

(1) 输入记录区,用来存放从输入文件读入的记录。再强调一次,一个输入文件只有一个输入记录区,用来放置当前记录。即使一个输入文件 A 中包含 1000 个记录,而内存的输入记录区中只有一个 A 文件的记录区,读入一个记录后,就把这个记录放到当前记录区,而把上次读入的记录从此区中"冲掉"。

如果有两个输入文件 A,B,则相应地开辟两个输入记录区,它们的大小根据文件节中的描述而确定,如有一输入文件已在文件节中定义:

```
FD INFILE LABEL RECORD IS STANDARD.
01   INREC.
  02  PRODUCT-NUM     PIC  9(6).
  02  PRODUCT-NAME    PIC  X(10).
  02  UNIT-PRICE      PIC  9(8).
```

当执行一个 READ 语句后,从 INFILE 磁盘文件中读入一组数据,则输入记录区的情况如图 4.4 所示,为每一个数据项开辟一个内存域,如 PRODUCT-NUM 的描述为 PIC 9(6),则开辟 6 个字节,将输入的 001010 六个数字存放在这里。

(2) 输出记录区。用来存放准备输出到输出文件的记录的。假如有一输出文件 OUT-FILE,在数据部的文件节中已定义如下:

```
FD   OUTFILE   LABEL RECORD IS   STANDARD.
01   OUT-REC.
  02  PRODUCT-NUM     PIC  9(6).
  02  PRODUCT-NAME    PIC  X(10).
  02  UNIT-PRICE      PIC  9(8).
```

02 AMOUNT PIC 9(8).

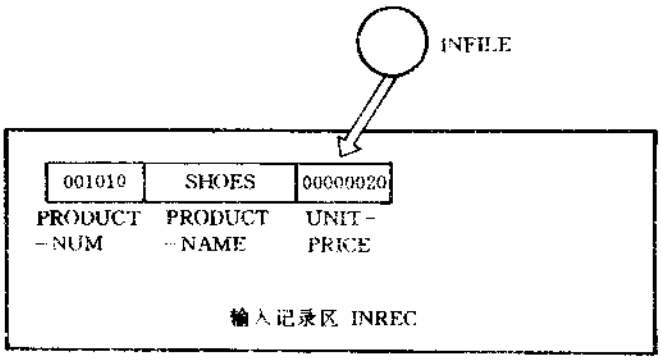


图 4.3

并已将有关的值传送到 OUT-REC 中，情况如图 4.4。

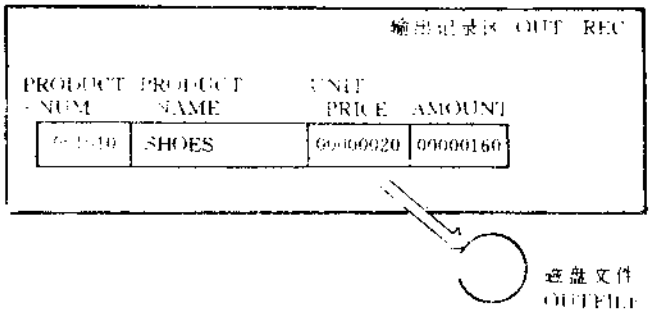


图 4.4

当执行一个 WRITE OUT-REC 语句时，就将此输出记录区中的数据写到 OUTFILE 中去。注意，只把数据写到 OUTFILE 中，而不是把数据名也写到输出文件中。

数据名只是在内存中作为数据项的标识用。如：MOVE AMOUNT TO AREA。就从名为 AMOUNT 的内存域中把它的值找出来传送到 AREA。但不能指定从磁盘上把名为 PRODUCT-NUM 的数据取出来。因为磁盘上只有数据，没有数据名。只有把这些数据读入内存相应的输入记录区以后，它们才和数据名发生联系。

(3) 工作单元区。一般来说，它用来存放程序中既不属于输入记录又不属于输出记录的数据，即所谓中间数据。在 WORKING-STORAGE SECTION(工作单元节)中定义的数据项都存放在这个区中。

假如有：

```
WORKING-STORAGE SECTION.
01  WORK REC.
  02  QUANTITY  PIC  9(3)  VALUE  8.
  02  CLASS  PIC  X  VALUE  'A'.
```

在工作单元区相应地开辟 WORK-REC 区域，存放它的值。见图 4.5。

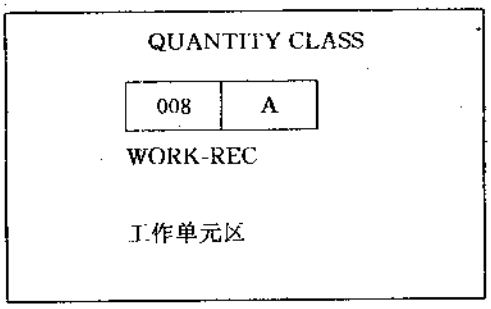


图 4.5

假如我们将上面几个部分联起来，写出下面的程序片断：

```
OPEN  INPUT  INFILE
      OUTPUT  OUTFILE.
⋮
READ  INFILE  AT  END  CLOSE  INFILE  OUTFILE
                       STOP  RUN.
MOVE  INREC  TO  OUT-REC.
COMPUTE  AMOUNT = QUANTITY * UNIT-PRICE  OF  INREC.
WRITE  OUT-REC.
⋮
```

图 4.6 表示此程序对数据的操作的情况(包括读、传送、写)。从图中也可以看到不同的数据存放在内存区的不同位置。

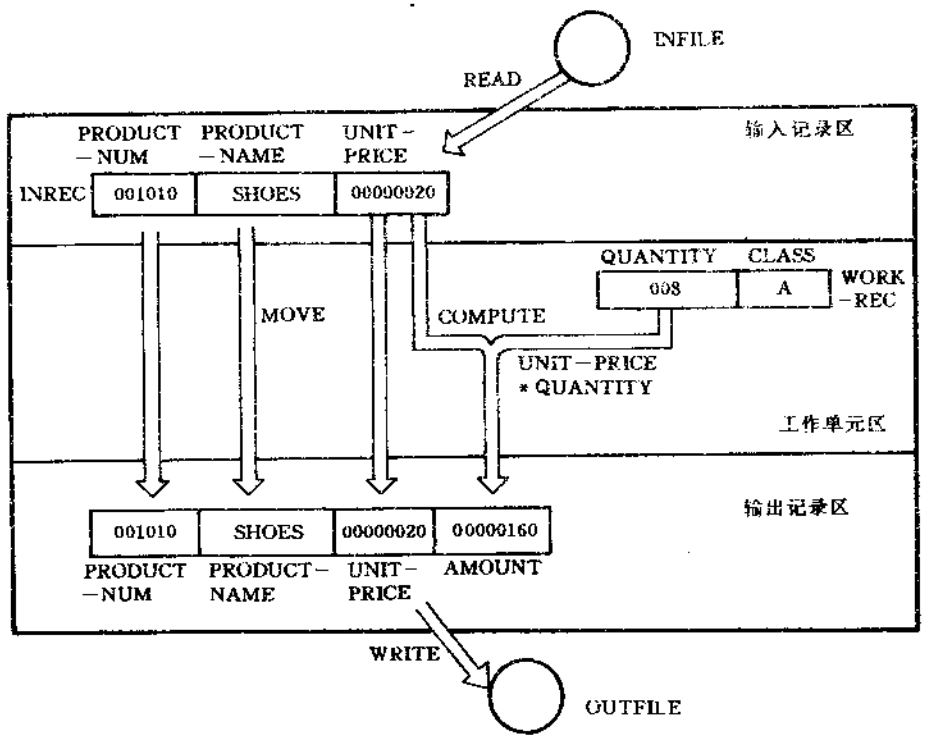


图 4.6

由此可见，区域图的作用是：

1. 明确表示输入记录区、输出记录区和工作单元区的不同性质、不同位置、不同用途。

2. 把在数据部中描述的各数据项形象地表示出来，比较直观，容易理解。

3. 对过程部中有关语句的作用作了形象的表示(如 READ，MOVE，WRITE 的操作，见图 4.6 中的箭头)。

4. 把数据部、过程部中有关成份结合在一起用图表示，使人们对数据部和过程部之间的关系有一个统一的概念。如果没有区域图，初学者往往对数据部的描述感到难以理解，对过程部与数据部的关系搞不清楚。因此，建议读者学会画区域图，它能帮助我们很快地理解别人的程序，也有助于我们正确地进行程序设计。

设计一个 COBOL 程序的步骤为：

(1) 分析题意。

(2) 画出流程图。

(3) 画出区域图。

(4) 编写程序。

§4.6 程 序 举 例

【例 4.5】 有一批售货情况的记录，其数据形式为：

H期	产品代码	顾客代码	数 量	单 价
9(6)	9(4)	9(4)	9(6)	9(6)

要求读入售货记录，并将其内容按如下要求输出到售货清单磁盘文件中。

空格	日期	空格	产品代码	空格	顾客代码	空格	数量	空格	单价
X(5)	9(6)	X(2)	9(4)	X(2)	9(4)	X(2)	9(6)	X(2)	9(6)

程序的 N-S 流程图见图 4.7。

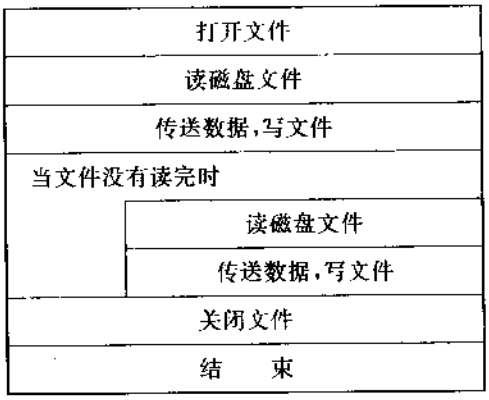


图 4.7

指定各数据项的名字并画出区域图，见图 4.8。

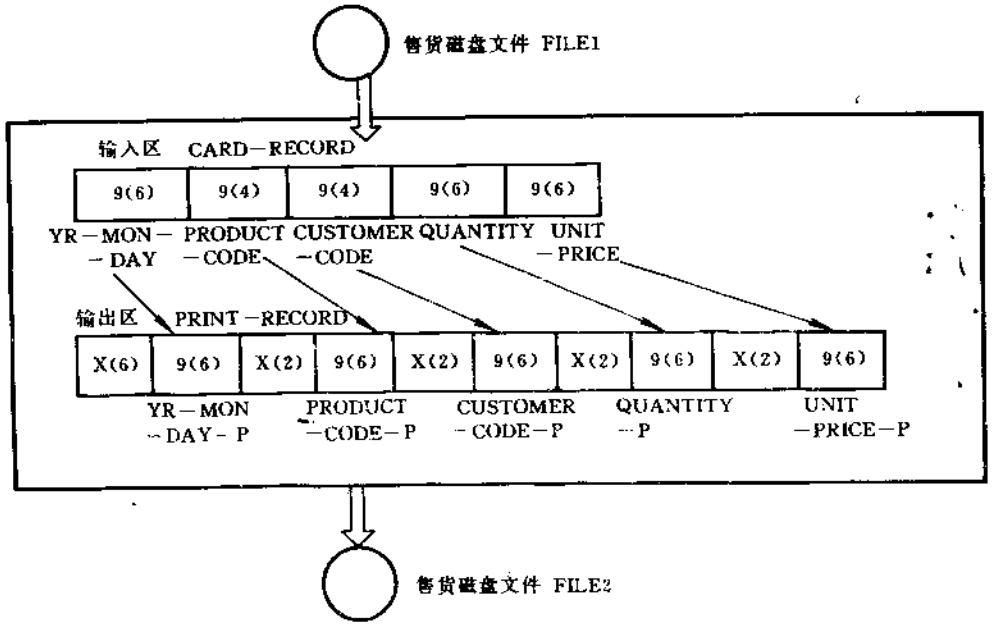


图 4.8

根据流程图和区域图写出程序如下：

```
IDENTIFICATION      DIVISION.                    (标识部)
PROGRAM ID. EXAM45.
ENVIRONMENT         DIVISION.                    (环境部)
INPUT-OUTPUT  SECTION.                           (输入输出节)
FILE-CONTROL.                                    (文件控制段)
    SELECT  IN-FILE       ASSIGN TO FILE1.
    SELECT  PRINT-FILE    ASSIGN TO FILE2.
DATA      DIVISION.                              (数据部)
FILE  SECTION.                                   (文件节)
FD  IN-FILE  LABEL  RECORD IS STANDARD.
01  IN-R.                                       (输入记录)
    02  YR-MON-DAY          PIC 9(6).               (日期)
    02  PRODUCT-CODE        PIC 9(4).               (产品代码)
    02  CUSTOMER-CODE       PIC 9(4).               (顾客代码)
    02  QUANTITY            PIC 9(6).               (数量)
    02  UNIT-PRICE          PIC 9(6).               (单价)
FD  PRINT-FILE  LABEL  RECORD IS STANDARD.
01  PRINT P.                                    (输出记录)
    02  FILLER              PIC X(6).
    02  YR-MON DAY-P        PIC 9(6).     (日期)
    02  FILLER              PIC X(2).
    02  PRODUCT-CODE-P      PIC 9(4).     (产品代码)
    02  FILLER              PIC X(2).
    02  CUSTOMER-CODE P     PIC 9(4).     (顾客代码)
    02  FILLER              PIC X(2).
    02  QUANTITY-P          PIC 9(6).     (数量)
```

```
    02  FILLER                PIC X(2).
    02  UNIT-PRICE P          PIC 9(6).      (单价)
PROCEDURE    DIVISION.                      (过程部)
S.  OPEN  INPUT    IN-FILE
          OUTPUT   PRINT-FILE.
    MOVE SPACE  TO  PRINT P.
R.  READ  IN-FILE
          AT END CLOSE IN-FILE,PRINT FILE
                 STOP  RUN.
M.  MOVE YR-MON-DAY        TO YR-MON DAY-P.
    MOVE PRODUCT-CODE      TO PRODUCT-CODE P.
    MOVE CUSTOMER CODE     TO CUSTOMER-CODE P.
    MOVE QUANTITY          TO QUANTITY-P.
    MOVE UNIT-PRICE        TO UNIT PRICE-P.
W.  WRITE  PRINT P  AFTER  2.
    GO TO R.
```

磁盘文件 FILE1 中记录数据形式如下：

```
92080100015512001000888888
92080400025513000155666666
92080700035521000250777777
          ⋮
```

磁盘文件 FILE2 中记录数据形式如下：

```
920801  0001  5512  001000  888888

920804  0002  5513  000155  666666

920807  0003  5521  000250  777777
```

程序说明：

(1) 有一个输入文件 IN FILE、一个输出文件 PRINT-FILE，它们分别和外部磁盘文件 FILE1 和 FILE2 相联系。

(2) 数据名的确定。一般 COBOL 程序设计者的习惯是，尽量使数据名具有明确的文字含意。即从英文(或汉语拼音)字中大致可以了解该数据项的含意。例如，用 YR-MON-DAY 表示"年-月-日"，一看便懂。这样起名的好处是使程序可读性好。比用符号(如 X、Y、Z…)代表一个数据项直观(当然读者也可以用其它便于理解的名字)。

在过程部为简化起见，以"S"代表"开始段"，以"R"代表"读数据"段，"M"代表"传送"段，"W"代表"写"段。这样选段名纯粹为方便。对本程序来说，M 段和 W 段的段名也可不写。即不必分这么多段。

(3) 每次将输入记录中的各数据项传送给输出记录区的相应的数据项。然后输出。

(4) 在输出时，由于输出记录的第一个字符作为走纸控制用。因此，在 PRINT-P 记录描述中，第一个初等项 FILLER 定为 X(6)，在输出时，实际只有 5 个空格。如果定为 X(5)，则只能输出 4 个空格了。

(5) 由于在执行 WRITE 语句后，输出记录区中的内容是不确定的，因此要将记录区的内容"冲空"这样才能保证 PRINT-P 中各 FILLER 项内容为空格。

(6) 本程序的输出结果送到磁盘文件 FILE2 中。如果需要打印出来，可以在程序运行结束后再用操作系统的打印命令通过打印机打印出来。这是中、小型机多用户系统常用的方

法，因为一个用户不能独占一台打印机，只能先放在磁盘上，需要时再打印。如果所使用的是微机或单用户系统的计算机，则可以在环境部中直接将 PRINT-FILE 与打印机联系，即“SELECT PRINT-FILE ASSIGN TO PRINTER”。(打印机名)，则程序的输出结果通过打印机输出。

【例 4.6】 有一批帐户的应收款的记录，形式如下：

ACCOUNT (帐号)	NAME (姓名)	AMOUNT (金额)
9(6)	X(10)	9(4)V99

要求：(1) 将金额大于 500 元的帐户金额和户名输出到磁盘文件中去。

(2) 计算出全部帐户的应收款额和每户平均款额。

程序如下：

```
IDENTIFICATION     DIVISION.          (标识部)
PROGRAM-ID. EXAM46.
ENVIRONMENT        DIVISION.          (环境部)
INPUT-OUTPUT   SECTION.
FILE-CONTROL.
    SELECT   IN-FILE   ASSIGN TO FILE1.
    SELECT   REPORT-FILE   ASSIGN TO FILE2.
DATA       DIVISION.                  (数据部)
FILE   SECTION.                       (文件节)
FD  IN-FILE   LABEL   RECORD   IS   STANDARD.
01  IN-R.
    02   ACCOUNT        PIC 9(6).          (帐号)
    02   NAME           PIC X(10).         (姓名)
    02   AMOUNT         PIC 9(4)V99.       (金额)
FD   REPORT-FILE   LABEL   RECORD IS STANDARD.
01   REP-P.
    02   FILLER         PIC X.
    02   REP-REC        PIC X(80).
WORKING-STORAGE   SECTION          (工作单元节)
77   COUNTER   PIC 9(6)V99   VALUE IS 0.
77   TOTAL     PIC 9(6)V99   VALUE IS 0.
01   DETAILS.
    02   ACCOUNT-D     PIC 9(6).          (帐号)
    02   FILLER        PIC X(3) VALUE IS SPACE.   (空格)
    02   NAME-D        PIC X(10).         (姓名)
    02   FILLER        PIC X(3) VALUE IS SPACE。   (空格)
    02   AMOUNT-D      PIC $(6). 99.      (金额)
01   SUMMARY.
    02   READER-1      PIC X(25)
         VALUE IS 'TOTAL ACCOUNTS RECEIVABLE'.
    02   FILLER        PIC X(3) VALUE IS SPACE.
    02   EDTOTAL       PIC $(4),$(3).99.
    02   FILLER        PIC X(3)   VALUE IS SPACE.
    02   HEADER-2      PIC X(15)
```

```
            VALUE IS 'AVERAGE EQUALS'.
        02  FILLER          PIC X(3) VALUE IS SPACE.
        02  AVERAGE         PIC $(2),$(3).99.
PROCEDURE       DIVISION.                (过程部)
K.  OPEN    INPUT       IN-FILE
            OUTPUT      REPORT-FILE.
    MOVE SPACE  TO  REP-P.
READING-DATA.
    READ  IN-FILE
            AT END GO TO WRAP-UP.
    IF AMOUNT >500.0
      MOVE  ACCOUNT   TO ACCOUNT-D
      MOVE  NAME      TO NAME-D
      MOVE  AMOUNT    TO AMOUNT-D
      MOVE  DETAILS   TO REP-P
      WRITE  REP P  AFTER 2.
    ADD  AMOUNT  TO  TOTAL.
    ADD  1  TO  COUNTER.
    GO  TO  READING-DATA.
WRAP-UP.
    MOVE  SPACE  TO  REP-P.
    MOVE  TOTAL  TO  EDTOTAL.
    DIVIDE  COUNTER  INTO  TOTAL.
    MOVE  TOTAL  TO  AVERAGE.
    MOVE  SUMMARY  TO  REP-REC.
    WRITE  REP-P  AFTER 2.
    CLOSE  IN-FILE  REPORT-FILE.
    STOP  RUN.
```

假设输入数据为以下六个记录：(输入文件为 FILE1.DAT)

```
000001WANG MIN          222277
000002LI HAI            673466
000003TIAN LI           009972
000004GAO HAI           423774
000005BAI WEI           234673
000006LI FUN            234567
```

运行后，结果输出到磁盘文件 FILE2 中。将它打印出来：如下：

```
000001    WANG  MIN     $2222.77

000002    LI  HAI       $6734.66

000004    GAO  HAI      $4237.74

000005    BAI  WEI      $2346.73

000006    LI  FUN       $2345.67

TOTAL ACCOUNTS RECEIVABLE  $17,987.29  AVERAGE  EQUALS $2,997.88
```

程序流程图如图 4.9 所示。

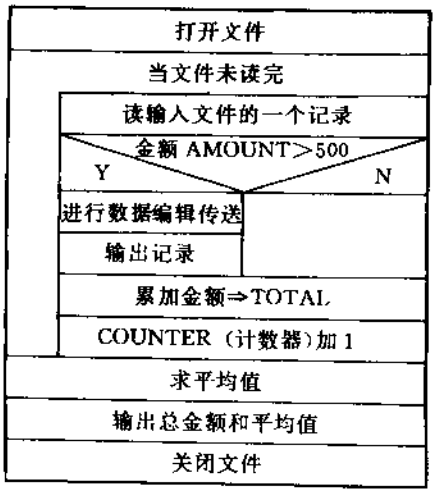


图 4.9

【例 4.7】 代销店在代销商品后，要向工厂索取代销费和手续费。月纯销售额（销售总数减去退货额）<5000 元时，代销费按 7.5%提取，外加手续费 100 元。纯销售额≥5000 时按 10%提取，外加手续费 150 元。

工厂要计算应付给各代销店的款数。输入数据为：代销人的编号、姓名、本月销售额，本月退货额。要求打印出：代销人编号、姓名、本月纯销售额，应付代理费（代销费加上手续费）。

输入的数据记录形式如下：

空	代销人号	空	代销人姓名	空	本月销售额	空	本月退款额
X(2)	X(4)	X(2)	X(10)	X(2)	9(5)V99	X(2)	9(4)V99

可画出流程图，见图 4.10。

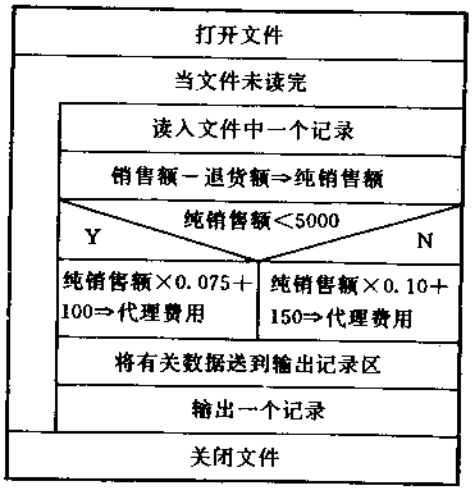


图 4.10

```
IDENTIFICATION      DIVISION.                    (标识部)
PROGRAM-ID. EXAM 47.
ENVIRONMENT         DIVISION.                    (环境部)
INPUT-OUTPUT  SECTION.                           (输入输出节)
FILE-CONTROL.                                    (文件控制段)
    SELECT  SALES-PERSON-FILE  ASSIGN TO FILE1.
    SELECT  REPORT-FILE  ASSIGN TO FILE2.
DATA      DIVISION.                              (数据部)
FILE  SECTION.                                   (文件节)
FD  SALES-PERSON-FILE
          LABEL  RECORD  IS  STANDARD
          DATA  RECORD  IS SALES-PERSON-RECORD.
01  SALES-PERSON-RECORD.                        (销售人记录)
    02  FILLER'        PIC X(2).
    02  SP-NUMBER     PIC X(4).                 (销售人号)
    02  FILLER         PIC X(2).
    02  SP-NAME        PIC X(10).               (销售人姓名)
    02  FILLER         PIC X(2).
    02  SP-CURRENT-SALE  PIC 9(5)V99.           (本月销售额)
    02  FILLER         PIC X(2).
    02  SP-CURRENT-RETURN  PIC 9(4)V99.(本月退货款)
    02  FILLER         PIC X(2).
FD  REPORT-FILE  LABEL  RECORD IS STANDARD
                 DATA RECORD  IS REPORT-RECORD.
01  REPORT-RECORD.                 (输出报表记录)
    02  FILLER         PIC X(10).
    02  RT-NUMBER     PIC X(4).
    02  FILLER         PIC X(6).
    02  RT-NAME        PIC X(10).
    02  FILLER         PIC X(6).
    02  RT-NET-CURRENT-SALE  PIC ZZ,ZZZ.99.  (纯销售额)
    02  FILLER         PIC X(16).
    02  RT-COMMISSION          PIC Z,ZZZ.99.  (代理费用)
WORKING-STORAGE  SECTION。     (工作单元节)
77  NET-CURRENT-SALES  PIC 9(5)V99.
77  COMMISSION             PIC 9(4)V99.
PROCEDURE      DIVISION.              (过程部)
OPEN-FILES.                           ("打开文件"段)
    OPEN  INPUT    SALES-PERSON-FILE
          OUTPUT  REPORT-FILE.
READ-INPUT.                           ("读入"段)
    READ  SALES-PERSON-FILE
          AT END CLOSE SALES-PERSON-FILE,REPORT-FILE
          STOP RUN.
CALCULATE-COMMISSON.                  ("计算代理费"段)
    SUBTRACT SP-CURRENT-RETURN FROM
    SP-CURRENT-SALE GIVING NET-CURRENT-SALES.
    IF  NET-CURRENT-SALES  IS  LESS  THAN  5000
        MULTIPLY NET CURRENT-SALES  BY  0.075
          GIVING COMMISSION
        ADD  100  TO  COMMISSION
ELSE
        MULTIPLY  NET-CURRENT-SALES  BY  0.10
        GIVING  COMMISSION
```

```
        ADD  150  TO  COMMISSION.
WRITE-OUTPUT.                              ("输出"段)
    MOVE  SPACE  TO  REPORT-RECORD.
    MOVE  SP-NUMBER  TO  RT-NUMBER.
    MOVE  SP-NAME  TO  RT-NAME.
    MOVE  NET-CURRENT-SALES  TO  RT-NET-CURRENT-SALE.
    MOVE  COMMISSION  TO  RT-COMMISSION.
    WRITE  REPORT-RECORD  AFTER  2.
    GO  TO  READ-INPUT.
```

设磁盘数据文件 FILE1 中有如下数据：

```
0001  CHANG  LI          1230012    050033
0002  MA  HONG           0050000    010000
0003  WANG  GONG         0120000    004000
0004  XIA  WEN           0200000    003000
0005  JIA  LI            0222222    036666
```

输出结果存放在磁盘数据文件 FILE2 中，其形式如下：

```
0001  CHANG  LI          11,799.79        1,329.97

0002  MA  HONG             400.00          130.00

0003  WANG  GONG         1,160.00          187.00

0004  XIA  WEN           1,970.00          247.75

0005  JIA  LI            1,855.56          239.16
```

程序说明：

本程序中用了两个中间数据项，用来存放计算中间结果（本月纯销售额 NET-CURRENT-SALES 和代理手续费 COMMISSION）。它们是在工作单元节中定义的。因为它们并不是属于输入或输出的数据项。

为了"成文自明"，在本程序中以 SALES-PERSON-FILE（代销人文件）来作输入文件名，以 REPORT-FILE（报表文件）来作为输出文件名。以 SALES-PERSON-RECORD 作输入记录名。在此记录中所有数据名都加上前缀 SP，表示是 SALES-PERSON-RECORD 中的项。以 REPORT-RECORD（报表记录）作输出的记录名，把其中的各项均加上前缀 RT，表示是 REPORT-RECORD 中的项。

程序的过程部并不复杂。有的初学者感到 COBOL 程序难懂并不在过程部中，而是对前面三个部分（特别是数据部）不太习惯。在有的程序中数据部的篇幅会占整个源程序的一半以上，初看起来使人抓不住重点。建议初学者看程序时先看过程部，画出流程图，对程序的思路有一清晰的概念。然后以过程部为线索去看数据部和环境部中的有关部分，画出区域图。就会感到条理清楚，层次分明，而不会感到是一堆乱麻，无从入手。

读者可以自己对程序作必要的补充，以打印出表头说明。

习　题

4.1　数据部的作用是什么？已学过的数据部有哪几节？每一节的作用是什么？

4.2　文件描述中应包含哪些必要的部分？记录名和文件名能否相同？记录描述体的层号是什么？

4.3　正确地填入数据项 A 在内存中的内容(任选一合法的值)。

数据项 A 描述符	99	X(5)	99.9	A(3)9	AX9AX	9(4)X(2)	99V99	99PPP
内存中内容								

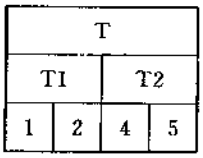
4.4　如果 A 是数值型数据，B 为编辑数值型数据，且按下表所示。如果执行 MOVE A TO B 后打印 B 内容，应得到什么结果？

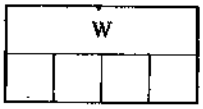
打印结果 / A 值 / A 描述 / B 描述	S 9(3)V99			
	+26.78	−68.75	+128.61	0
+999.99				
+99.9				
−9,999.99				
$9(4).9+				
$(5).$(2)−				
+ZZZ,ZZZ.99				
+(4).++				
−(4).−−				
$*(3).**CR				
*****.**				

4.5　PIC 子句使用的范围有什么限制？下面的写法有什么错误，请改正。

```
(1) 01   A     PIC  X(10).
         02   A1 PIC  X(2).
         02   A2 PIC  X(8).
(2) 01   B  PIC  X(8).
         02   B1.
         02   B2.
```

4.6　组合项在作为一个整体传送时，它是按什么类型处理的？以下的传送：MOVE T TO W，是否可以？





如可以，W 的值是什么？已知 T 和 W 的描述分别为：

```
77  W   PIC  X(4).
01  T.
    02  T1   PIC  99.
    02  T2   PIC  99.
```

4.7　下表给出数据项 A 的描述和它在内存中的内容，请写出在执行 DISPLAY A 语句时显示出的结果。(A 的内容指的是 A 中各字节中实际存放的字符)。

A 描 述	99V99	9V9	999	9999	99PPP	PPP99	X(4)	A(3)X(2)
A 的内容	9134	12	000	0010	18	17	1234	ABC12
显示出 A 的结果								

4.8　请写出执行 MOVE A TO B 后，B 的内容。

A　值	B 描 述	B 内 容
1234	99	
"ARRAY"	X(6)	
"HELLO"	X(3)	
116.87	99V99	
87.251	999	
0	999V999	

4.9　看懂下面程序。(1) 画出框图和区域图。(2) 设计输入文件上数据排列格式。(3) 画出输入记录区的数据组织形式。(4) 写出打印格式。(5) 注明每个部、节、段的名称和作用。

```
IDENTIFICATION      DIVISION.
PROGPAM-ID          EX.
ENVIRONMENT         DIVISION.
INPUT-OUTPUT        SECTION.
FILE-CONTROL.
    SELECT  IN-FILE  ASSIGN  TO  U01.
    SELECT  PRINT-FILE  ASSIGN  TO  U02.
DATA  DIVISION.
FILE    SECTION.
FD      IN-FILE
          LABEL  PECORO  IS  STANDARD.
          DATA  RECORD  IS  IN-RECORD.
01        IN-RECORD.
  02  NAME       PIC  X(15).
  02  STREET     PIC  X(25).
  02  CITY       PIC  X(40).
FD    PRINT-FILE LABEL RECORD IS STANDARD.
        DATA RECORD IS PRINTER-LINE.
```

```
01    PRINTER-LINE.
      02  FILLER  PIC  X.
      02  PRINT-LINE  PIC  X(136).
PROCEDURE  DIVISION.
OPENER.
      OPEN  INPUT  IN-FILE
      OUTPUT  PRINT-FILE.
      MOVE  SPACE  TO  PRINTER-LINE.
READ-PRINT
      READ  IN-FILE  AT  END  GO  TO  JOB-END.
      MOVE  NAME  TO  PRINT-LINE
      WRITE  PRINTER-LINE  BEFORE  1.
      MOVE  STREET  TO  PRINT-LINE.
      WRITE  PRINTER-LINE  BEFORE  1.
      MOVE  CITY  TO  PRINT-LINE.
      WRITE  PRINTER-LINE  BEFORE  4.
      GO  TO  READ-PRINT.
JOB END.
      CLOSE  IN-FILE  PRINT-FILE
      STOP  RUN.
```

4.10　把上面的程序改成用 ACCEPT 语句输入地址数据(包括姓名、街道、城市名)，当输入并输出完 5 组数据后，使程序结束运行。

4.11　学生成绩的记录，内容包括：学号、姓名、五门课成绩。将记录依次送入，要求输出一张全班学生成绩清单，除包括上述各项外，还要求计算出每个学生的平均成绩。

4.12　在上题的基础上，再补充一个要求：统计输入记录的数目(即学生数)，并在输出完每个学生成绩后，再输出全班人数和全班总平均成绩(不要求算每门课平均分数)。

第五章　过程部之二

至今为止，我们已经对 COBOL 源程序的四大部分有了一个初步的了解，可以着手编制一些简单的然而是完整的 COBOL 程序。在此基础上，我们从本章开始介绍 COBOL 程序设计的较高的技巧。也就是说，在编制比较复杂的程序时，已学过的知识还是不够的；需要进一步掌握 COBOL 中其它的功能。本章介绍过程部的较高技巧，使读者能用相对简单的语句来完成较复杂的数据处理任务。

§5.1　传送语句(MOVE 语句)的较高技巧

5.1.1　各种类型数据之间的传送

(一) 我们先简单回顾一下同类型数据间的传送规则：

数值型数据之间的传送，按小数点位置对齐，如发送项长于接收项，则多余位截去，如短于接收项，接收项的空位补零。

字母或字符型数据间的传送，按左端对齐，如发送项长于接收项，右端多余位截去，如短于接收项，右端补空格。

(二) 编辑传送。发送项是数值型数据，而接收项是编辑数值型数据，则先将发送项中数据按接收项的描述要求进行编辑，然后再传送。这种传送方式用得很普遍，希望读者能较好地掌握。注意：传送的方向是：由数值型数据传送给编辑型数据，而不能由编辑型数据传送给数值型数据。例如：

```
77  A   PIC   9(4)V99.
77  B   PIC   $(6).99.
MOVE   A   TO   B      是正确的.
MOVE   B   TO   A      是错误的.
```

如果 A 的内容原为 $876\overset{\wedge}{3}65$，执行 MOVE A TO B 后，B 的内容为 $8763.65。显然，再将 B 传送回 A 是做不到的。

(三) 不同类型数据间传送的规则：

见表 5.1

说明：表中"√"为允许的传送，"×"为不允许，"△"为在某些情况下是正确的，即有条件的。如字符型数据向整数数据项传送，如字符型数据内容全为数字则为正确，如有非数字则不正确。又如字符型数据向字母型数据项传送，若 A，B 的描述分别为：

```
77  A   PIC   X(4)   VALUE 'AB8D'.
77  B   PIC   A(4).
```

当执行 MOVE A TO B 时，由于 A 的值"AB8D"不是字母型的(其中有一个为"8")，所以是不正确的。而如果 A 的值为"ABCD"，则正确，传送后 B 值为"ABCD"。因此，只有在一定条件下由字符型数据传送到字母型数据项才是正确的。

表 5.1

发送项 \ 接收项		数值型		数值编辑型	字母型	字符型	字符编辑型	组合项
		整数	非整数					
数值型	整数	√	√	√	×	√	√	√
	非整数	√	√	√	×	×	×	√
编辑数值型		×	×	×	×	√	√	√
字母型		×	×	×	√	√	√	√
字符型		△	△	△	△	√	√	√
编辑字符型		×	×	×	×	√	√	√
数值常量		√	√	√	×	×	×	√
非数值常量		×	×	×	√	√	√	√
ZERO		√	√	√	×	√	√	√
SPACE		×	×	×	√	√	√	√
组合项		△	△	△	△	√	√	√

不同类型数据间传送的规则可以归纳如下：

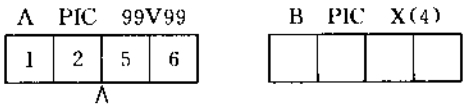
(一) 非法的传送：

(1) 数值编辑项、字符编辑项、SPACE，字母数据项不能传送给数值数据项或数值编辑项。例如：A 和 B 都是用 PIC 9(3).99 描述的，MOVE A TO B 是不允许的。

(2) 数值常量、ZERO、数值数据项、数值编辑项不能传送给字母数据项。

(3) 非整数的数值数据项(如用 PIC 99V99 描述的)或数值常量(如不带引号的数值)不能传送给字符数据项或字符编辑数据项。

例如：

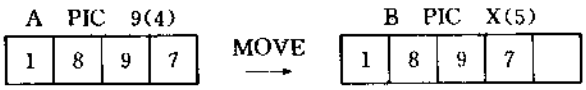


MOVE A TO B 是非法的。

(二) 合法的传送：

(1) 接收项为字符数据项或字符编辑项，而发送项的长度大于接收项的，按“对齐”的原则(一般为左对齐，当有 JUSTIFIED 子句时为右对齐)，超过部分截断，如长度小于接收项的，多余位置补空格。

如：



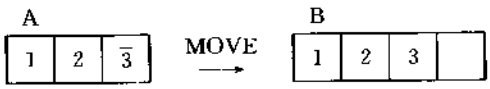
当发送项的描述是带符号的数值(如用 PIC S99 描述的)时,符号不予传送。

如77 A PIC S999 VALUE −123.

77 B PIC X(4).

A 的内容为"1 2 $\bar{3}$",占三个字节。若执行:

MOVE A TO B.

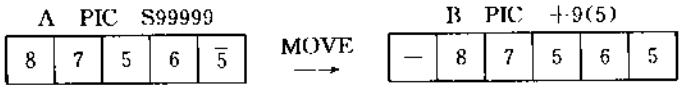


因为符号不传送,而字符项的接收为"左对齐",故右补一空格。

(2) 接收项是数值项或数值编辑项(初等项),可以接收数值型的数据以及内容为全数字的字符型数据项。

在接收数值型数据时,按小数点位置对齐,多余位置补零(如果在编辑符中要求置换前导零者除外,如用 $ (9)99)。

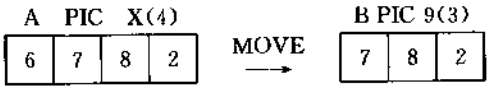
如接收项是带符号的数值项(如用 PIC S99999 描述的),将发送项的符号接要求传送。



如接收项是不带符号的数值项,按发送项的绝对值传送,即在接收项中不出现符号。

如发送项是字符型数据项,而其内容全为数字,则按无符号整数传送。

传送后 B 的值为 8751,是数值型数据,可以参加运算,例如:在执行 ADD 10 TO B 后,B 的值变成 8761。如果接收项长度不够长,则发出截断。如:



整数传送时按右对齐,左截断。如果传送小数,则按小数点对齐原则。如:

7 5 两个数字被截断，B 可以用于运算，值为 13.00。

不接受非数值常量。当发送项是字符型数据项，而其内容为非数字字符，则会出现不可预测的结果（各计算机系统处理的方法不同）。

（3）接收项是字母型，按左对齐原则接收字母字符，多余位置补空格。但它不应接收非字母的字符。

注意，发送项和接收项在内存中不能有重叠的部分。例如：

```
02  A.
  03  B  PIC  99.
  03  C  PIC  999.
```

执行 MOVE A TO B 或 MOVE B TO A 也是不行的，结果不可预测。

在程序中，最常见的传送有：

（1）同类型数据的传送，如数值型数据项之间（包括整数型数据项与非整数型数据项之间的传送），字符型数据项之间的传送。

（2）数值型数据项向编辑数值型数据项的传送，以便在编辑后输出。

（3）各类型数据项（除了非整数数值型数据项，如 99V99 类型）可向字符型数据项传送。从表 5.1 中可以看出，字符型数据项能够接纳除了非整数数值型数据项以外的所有类型数据项。例如：

```
77  A  PIC  $99.99.
77  B  PIC  X(6).
```

如果 A 的内容原为 $18.75，则执行 MOVE A TO B 后，B 的内容也是 $18.75。

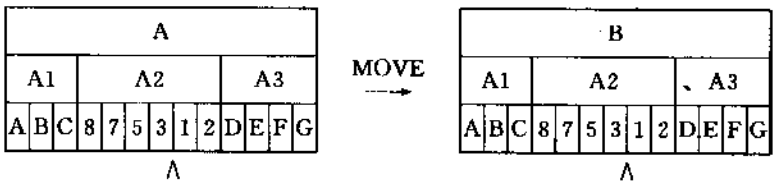
5.1.2 组合项的传送

可以一次传送一个初等项，也可以传送一个组合项或一个记录。组合项的传送是将发送项的内容不加转换地一个字节一个字节地顺序传送到接收项。

（一）发送项和接收项都是组合项，而且其结构和描述均相同。则可看作将各初等项一一对应传送。如：

```
01  A.                          |  01  B.
  02  A1  PIC  X(3).            |    02  B1  PIC  X(3).
  02  A2  PIC  9(4)V99.         |    02  B2  PIC  9(4)V99.
  02  A3  PIC  A(4).            |    02  B3  PIC  A(4).
```

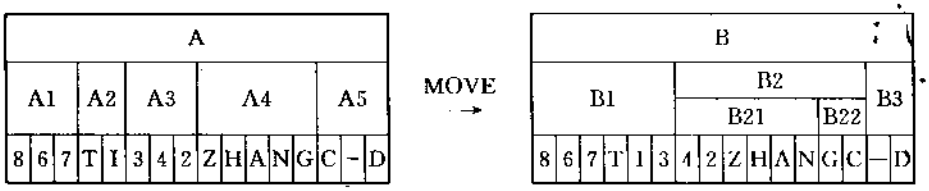
传送后，A 和 B 的各初等项内容完全相同。



（二）如发送项与接收项长度相同，但数据结构形式不同，则将发送项的内容原样不变

地自左而右顺序地传送到接收项。如：

```
01  A.                          |   01  B.
    03  A1  PIC  999.           |       04  B1  PIC  X(6).
    03  A2  PIC  X(2).          |       04  B2.
    03  A3  PIC  999.           |         05  B21  PIC  X(6).
    03  A4  PIC  A(5).          |         05  B22  PIC  A(2).
    03  A5  PIC  X(3).          |       04  B3  PIC  X(2).
```



它不是按初等项传送的，而将各字节内容自左而右一一传送过去。显然，B1，B21 都不是数值型数据（尽管它们包括了数字）。

它等价于在数据部中定义数据项 A 和 B：

```
01  A  PIC  X(16).
01  B  PIC  X(16).
```

在过程部中用以下语句将数据项 A 的内容传送给 B。

```
MOVE A TO B.
```

需要强调的是：组合项作为一个整体而言，它决不会是数值型的，在一般情况下可把它看作是字符型，如：

A 和 B 为组合项，它们下属的初等项都是数值型的。但 A 和 B 本身并不是数值型的，A 和 B 的值并不等于数值 122436。在将 A 和 B 传送时，是按内存中实际存放的信息一个字符一个字符传送的。

如果传送时，发送项与接收项长度不同，按已介绍过的规则：左对齐，右补空格，多余位截去。

5.1.3 对应传送（带 CORRESPONDING 子句的 MOVE 语句）

（一）数据名的受限和受限名的传送

在前面几章介绍的程序中，我们尽量不在同一个 COBOL 源程序中使用相同的数据名。但是，在比较复杂的 COBOL 程序中往往使用同一个数据名来代表不同的数据项。如：

```
02  SUM.
    04  A1  PIC  X(2).
```

```
    04  A2  PIC  X(3).
    04  A3  PIC  9(4).
  ⋮
02  TOTAL.
    05  A1  PIC  X(4).
    05  A3  PIC  9(5).
    05  B2  PIC  99V99.
```

A1 和 A3 都出现了两次。每个 A1 都有自己的描述，代表一个数据项。但如何区别它们呢？如果有 MOVE A1 TO T 语句，则无法执行。因为 A1 这个数据名不是唯一的。应解决这个矛盾，否则将发生混乱。正如在一个学校中有好几个名字为李建国的学生。校长点名时，如只喊名字"李建国"，则好几个学生会同时举手，这时校长应在名字前加以"限定词"，即"高一甲班的李建国"或"初二乙班的李建国"，这就区别开了。如果该班有两个李建国，则再加一层限定词，如"初二乙班第一小组的李建国"。

在 COBOL 中，也用这一方法，如：

```
MOVE  A1  OF  SUM  TO  T1.
MOVE  A1  OF  TOTAL  TO  T2.
```

即指出将"组合项 SUM 中的 A1"传送给 T1，将"组合项 TOTAL 中的 A1"传送给 T2。"A1 OF SUM"和"A1 OF TOTAL"就变成唯一的数据名了。也就是说，用上属层的数据名来对一个下属层的数据名加以"限定"，以使其成为唯一。这种方法，称为数据名的受限。在上例中称 SUM 和 TOTAL(较高层的数据名)为限定符。相当于"初二乙的×××"，初二乙就是限定符，用来限定范围的。数据名和限定符之间用 OF 或 IN 来连接，表示"SUM 里面的 A1"或"SUM 的 A1"。如果限定一次还不能成为唯一，可用多次限定符。假如上例的描述中增加了一个 03 层，即：

```
02  SUM.
  03  A.
    04  A1  PIC  X(2).
    04  A2  PIC  X(3).
    04  A3  PIC  9(4).
02  TOTAL.
  03  A.
    05  A1  PIC  X(4).
    05  A3  PIC  9(5).
    05  B2  PIC  99V99.
```

这时，如果只用一个限定符，如：A1 OF A 还不能区别两个 A1，因它们的上属层也同名，因此还要上溯到不同名的层为止，即：

```
MOVE  A1  OF  A  OF  SUM  TO  T1.
MOVE  A1  OF  A  OF  TOTAL  TO  T2.
```

数据名在受限后构成的一个整体(如 A1 OF A OF TOTAL 和 A1 OF A OF SUM)称为受限名。受限名是唯一的，因而是标识符的一种形式。

程序中为什么要采用同一个数据名来代表不同的项？这是由于使程序行文清楚，增加程序的可读性和清晰性。如：

```
01  OLD-RECORD.  (老记录)
  02  TODAY-DATE (今日日期)
    03  MONTH …
    03  DAYY …
    03  YEAR …
  02  LAST-PERIODS-DATE. (最近一次的日期)
    03  MONTH …
    03  DAYY …
    03  YEAR …
    03  TOTAL …
```

这里，都用了 MONTH，DAYY，YEAR 代表月、日、年(因 DAY 是保留字，故用 DAYY 代表'日')。当然也可以用 A1，A2，A3 和 B1，B2，B3，但影响了程序的可读性和清晰性。用：

```
MOVE MONTH OF TODAY-DATE TO MONTH OF LAST-PERIODS-DATE.
MOVE DAYY OF TODAY-DATE TO
          DAYY OF LAST-PERIODS-DATE.
MOVE YEAR OF TODAY-DATE TO
          YEAR OF LAST-PERIODS-DATE.
```

显然意思比 MOVE A1 TO B1 清楚。

(二) 用 CORRESPONDING 子句的传送——对应传送(同名传送)

用受限名传送是可行的，但要一一写出各受限名，如上面列出的那样，用三个 MOVE 语句才能将年、月、日传送到另一组合项中去，程序比较长。COBOL 提供了"对应传送"的功能，即把一个组合项中若干项传送给另一组合项中同名的项。如上例中，我们可以只用一个语句：

```
MOVE CORRESPONDING  TODAY-DATE TO  LAST PERIODS-DATE.
```

则把组合项 TODAY-DATE 中的 MONTH，DAYY，YEAR 传送给组合项 LAST-PERIODS-DATE 中的 MONTH，DAYY，YEAR。因为在 TODAY-DATE 与 LAST-PERIODS-DATE 中都具有同名的 MONTH，DAYY，YEAR 项。这就是对应传送或同名传送。其一般格式为：

```
MOVE { CORRESPONDING } 标识符1  TO  标识符2
     { CORR          }
```

说明：(1)如果两个组合项中包括的项不同，则只传送同名的项。如有以下两个组合项：

```
01  PAY-RECORD.                    01  EDITTED-RECORD.
  02  A1  PIC  9(4)V99.              02  B3  PIC  ZZZZ.99.
  02  A2  PIC  9(4)V99.  ------->    02  A2  PIC  ZZZ9.99.
  02  A3  PIC  9(3)V99               02  A1  PIC  ZZZ9.99.
```

如果执行 MOVE CORR PAY-RECORD TO EDITTED-RECORD，只传送同名的项，即相当于：

```
MOVE  A1  OF  PAY RECORD  TO  A1  OF  EDITTED-RECORD.
MOVE  A2  OF  PAY-RECORD  TO  A2  OF  EDITTED RECORD.
```

而 A3并不传送给 B3。

如果两个组合项的结构和描述完全相同。如：

```
01  A.                          01  B.
  02  A1  PIC  X(3).              02  A1  PIC  X(3).
  02  A2  PIC  X(4).              02  A2  PIC  X(4).
```

则：MOVE A TO B

和：MOVE CORR A TO B 的作用完全相同。

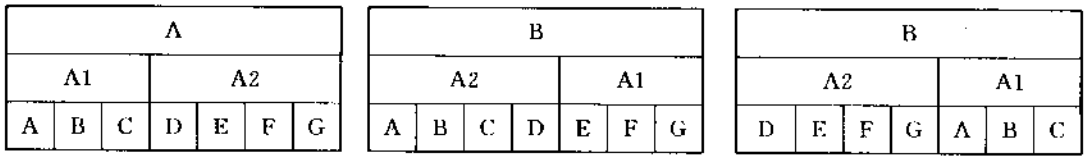
CORR 是 CORRESPONDING 的缩写，二者等价。

但如果两个组合项的结构不同，如：

```
01  A.                          01  B.
  02  A1  PIC  X(3).              02  A2  PIC  X(4).
  02  A2  PIC  X(4).              02  A1  PIC  X(3).
```

假如原来 A1的内容为 ABC，A2的内容为 DEFG，则：

MOVE A TO B 和 MOVE CORR A TO B 的结果不同。见下图：



（执行 MOVE A TO B 后）　（执行 MOVE CORR A TO B 后）

即对应传送是分别按各对应的项传送，而组合项传送则将 A 的内容按字符次序一一传送到 B，而不问 A1与 A2的次序如何。

（2）传送的两者间必须有成对的同名数据项，而且这一对中必须至少有一个项是初等项。否则不能作为对应项传送。如：

```
01  A.                          | 01  A1.
  02  B  PIC  X(2).             |   03  B  PIC  X(2).
  02  C.                        |   03  C.
    03  C1  PIC  X(4).          |     05  C3  PIC  X(6).
    03  C2  PIC  X(5).          |     05  C4  PIC  X(3).
```

如果有一个 MOVE CORR A TO A1语句，是如何传送的呢？

在 A 和 A1中，B 是同名的数据项（只要求同名，并不要求层号也相同，如例中一个 B 的层号为02，一个 B 的层号为03，是允许的。它们虽具有不同层号，但都是 A 或 A1的下一层），将 B OF A 传送到 B OF A1。再看 C 也同名，但这两个 C 中没有一个是初等项，而且它们的下属项也不同名，因而不能作为对应传送的对象。即不会执行 MOVE C OF A TO C OF A1。

如果将 A1的结构改为：

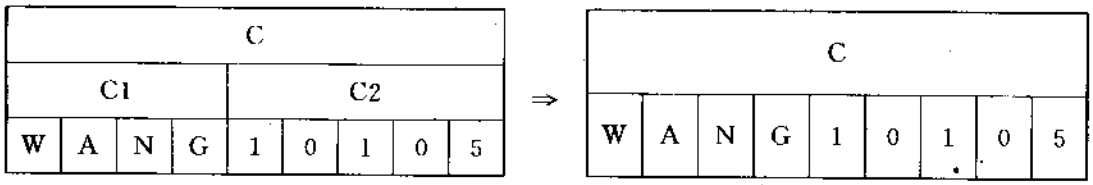
```
01  A1.
  03  B  PIC  X(2).
  03  C  PIC  X(9).
```

即 C 为初等项，则可以按对应传送。执行：

MOVE CORR A TO A1 相当于：

MOVE B OF A TO B OF A1.

MOVE C OF A TO C OF A1.

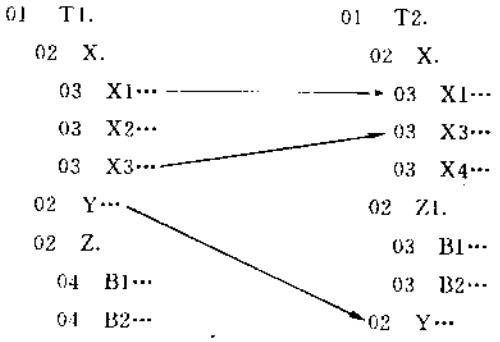


(3) 所谓同名，指的是它们有相同的全程受限，或简单地说，全程同名(当然不要求MOVE语句中出现的两个标识符同名)。

如果要执行：

MOVE CORR T1 TO T2.

假如T1和T2的描述分别为：



其中X1，X3是同名数据项，它们之上还有一个X，也是同名的，因此，X1和X3是两个组合项T1和T2中全程受限情况相同的项(因为除了MOVE中出现的标识符T1和T2名以外，二者在全程中名字完全相同)，即“全程同名”，可作对应项传送。Y也是对应项。而两个组合项中虽然也有同名的B1和B2，但它们的上属项不同名。它们分别属于Z和Z1，也就是说，它们受限情况不同，即B1 OF Z和B1 OF Z1不相同，或者说，它们不是全程同名。因此不是对应项，不按对应项传送。

上述 MOVE CORR T1 TO T2.

相当于：

```
MOVE X1 OF X OF T1 TO X1 OF X OF T2
MOVE X3 OF X OF T1 TO X3 OF X OF T2
MOVE Y OF T1 TO Y OF T2.
```

*(4) 在MOVE CORR A TO B中，A和B为组合项，而在属于A和B的各数据项中，那些带有RENAMES子句(重命名子句)或REDEFINES子句(重定义子句)或OCC URS子句(重现子句)的数据项将不予以传送。如果在数据项A和B的描述中包含REDEFI NES子句或OCCURS子句(或从属于含有这些子句的数据项)，则在传送语句中，应当使用表元

素名，如 MOVE CORR A (1) TO B(1)。有关这些子句的含义和用法将在第七章和第九章(下册)中介绍。在此处先提醒一下，以便学完全书后在用到 MOVE 语句时有所遵循。

(三) 举例

【例 5.1】 将本书第四章例4.4改用对应传送处理。

```
IDENTIFICATION          DIVISION.
PROGRAM-ID.   EXAM51.
ENVIRONMENT             DIVISION.
INPUT OUTPUT   SECTION.
FILE-CONTROL.
       SELECT   INPUT-FILE     ASSIGN TO IN-FILE.
       SELECT   PRINT FILE     ASSIGN TO PRI-FILE.
DATA        DIVISION.
FILE     SECTION.
FD     INPUT FILE   LABEL   RECORD IS STANDARD.
01 GZQD-R.
       02   BH          PIC 9(6).          (编号)
       02   XM          PIC X(10).         (姓名)
       02   JBGZ        PIC 9(3)V99.       (基本工资)
       02   FJGZ        PIC 9V99.          (附加工资)
       02   FSBZ        PIC 9V9.           (副食补助)
       02   XLF         PIC 9V99.          (洗理费)
       02   TEF         PIC 99V99.         (托儿费)
       02   FZ          PIC 99V99.         (房租)
       02   HZJ         PIC 99V99.         (互助金)
       02   BSJKC       PIC 99V99.         (病事假扣除)
FD     PRINT FILE   LABEL   RECORD   IS   STANDARD.
01   GZQD P.
       02   FILLER     PIC X.
       02   BH         PIC 9(6).
       02   FILLER     PIC X.
       02   XM         PIC X(10).
       02   FILLER     PIC XX.
       02   JBGZ       PIC 9(3).99.
       02   FILLER     PIC XX.
       02   FJGZ       PIC 9.99.
       02   FILLER     PIC XX.
       02   FSBZ       PIC 9.9.
       02   FILLER     PIC XX.
       02   XLF        PIC 9.99
       02   FILLER     PIC XX.
       02   TEF        PIC 99.99.
       02   FILLER     PIC XX.
       02   FZ         PIC 99.99.
       02   FILLER     PIC XX.
       02   HZJ        PIC 99.99.
       02   FILLER     PIC XX.
       02   BSJKC      PIC 99.99.
       02   FILLER     PIC XX.
       02   SFGZ       PIC 9(4).99
```

```
PROCEDURE          DIVISION.
K.    OPEN  INPUT        INPUT-FILE
            OUTPUT       PRINT FILE.
D.    MOVE  SPACE  TO  GZQD-P.
      READ  INPUT-FILE
        AT END CLOSE INPUT-FILE,PRINT-FILE
               STOP  RUN.
S.    MOVE CORR GZQD-R TO GZQD-P.
SU.   COMPUTE SFGZ = JBGZ OF GZQD-R + FJGZ OF GZQD-R + FSBZ OF GZQD-R
                   + XLF OF GZQD-R-TEF OF GZQD R-FZ OF GZQD R
                   - HZJ OF GZQD-R BSJKC OF GZQD R.
X.    WRITE GZQD P AFTER 2.
      GO TO D.
```

显然，这个程序比以前的程序简练多了，在S段只用了一个MOVE语句，而例4.4的程序却用了十个MOVE语句。但应注意，由于存在相同的数据名，因此在SU段的计算语句中，对数据名应加以必要的限定。本程序在数据部描述输出数据项时，用Z编辑符来取消打印时的前零，读者可将它和前面的程序比较一下。

§5.2 算术运算语句的较高技巧

在这一节中将介绍在算术运算语句中使用ROUNDED子句，ON SIZE ERROR子句，CORRESPONDING子句和除法语句中的REMAINDER子句。过程部中的子句(Clause)是语句中的一个附属的部分，在COBOL中这个附属的部分一般是可以任选的(可选用或不选用)。

5.2.1 四舍五入处理(ROUNDED子句)

在加、减、乘、除和计算语句中，计算结果可能比接收项允许的长度长(包括整个数字长度，或者整数部分长度，或小数部分的长度)，则发生截断。见下表：

计算结果	接收项描述	接收项内容	说　明
86.7851	9V9	$\overset{67}{\wedge}$	整个长度不够
751.187	99V999	$\overset{51187}{\wedge}$	整数部分不够
12.34	9(5)V9	$\overset{000123}{\wedge}$	小数部分不够
8765.4321	9(3)V9(6)	$\overset{765432100}{\wedge}$	整数部分不够

关于整数部分由于长度不够而截断的情况，我们在本章§5.2.2节中讨论。现在先讨论小数部分的截断。

如果计算结果为186.7851，当数据项的描述不同时，有不同的截断情况，即近似情况不同(见下表)：

接收项描述	接收项内存内容
999	186
999V9	1867
999V99	18678
999V999	186785
9(3)V9(4)	1867851

为了提高精确度，一方面可以适当增加接收项小数点后的位数，同时可以采用四舍五入方法。用 ROUNDED 就表示将截断后的一位按四舍五入处理。如果有：

ADD A,B TO C ROUNDED

表示将 A+B+C 的值放入 C 中，要四舍五入(见下表)。

A+B+C 值	C 描 述	有无 ROUNDED	C 内 容
186.7851	999	有	187
186.7851	999V9	有	1868
186.7851	999V99	有	18679
186.7851	999V999	有	186785

可以看到：ROUNDED 的作用是，按照数据项的描述要求对多余位截断，然后对被截断的后一位数进行四舍五入处理。显然，如果给的描述不同，则截断的位置就不同，需要四舍五入的位也不同。

如果计算结果有多个，则应分别说明哪一个接收项要进行舍入处理，ROUNEDE 应写在有关的接收项(结果数据项)的数据名后面。如：

ADD A, B, C TO D, E, F ROUNDED, G ROUNDED, H

若 D,E,F,G 和 H 在加上(A+B+C)之后的值均为1283.657，则各数据项中的值见下表：

数据名	描 述	数据项的值	说 明
D	9999V9	12836	无舍入，截断
E	9999V99	128365	无舍入，截断

（续表）

数据名	描　述	数据项的值	说　明
F	9999V99	128366	舍入
G	9999V9	12837	舍入
H	9999V999	1283657	无舍入，截断

5.2.2 长度溢出处理

计算结果的整数部分的长度如果比结果数据项描述所规定的整数部分长，则称长度溢出。

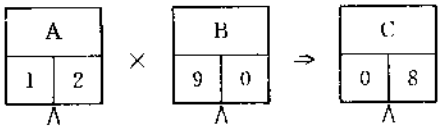
如：

```
MULTIPLY  A  BY  B  GIVING  C.
```

如有：

```
77  A  PIC  9V9  VALUE  1.2.
77  B  PIC  9V9  VALUE  9.0.
77  C  PIC  9V9.
```

A 和 B 的乘积应为10.8，将此结果放在 C 中，但 C 用9V9描述，放不下二位整数，高位截断（见下图）。



C 的内容为08。显然与应有的结果相差太大，虽然程序仍能断续运行，但算出的结果将是无意义的，错误会愈来愈大。应作专门溢出处理，表示“结果放不下，另作特殊处理”。

有些读者可能会想，为什么小数点后的截断不按长度溢出处理，只对整数部分的高位截断进行溢出处理？因为小数部分的截断只影响数值的近似程度，而不致影响数值的本质。如3.1415928，3.14159，3.14都是 π 的近似值，只是近似程度不同。而1234.567的高位如被截去，就变成234.567或34.567了，产生数倍、数十倍甚至更多的误差，即计算结果完全错了，不应继续运算下去。

ON SIZE ERROR 子句提供“溢出处理”。即当发生溢出错误时，按程序设计者事先指定的操作处理。如：

```
MULTIPLY A  BY  B  GIVING  C
        ON SIZE ERROR DISPLAY 'SIZE ERROR'
                STOP RUN.
```

这个语句的意思是：A×B⇒C，当 C 发生长度溢出错误时，显示出 SIZE ERROR 字样，然后停止运行，否则继续运行下去。

ON SIZE ERROR 子句实际上相当一个条件语句，即如果满足溢出条件，则执行 ON

SIZE ERROR 后面的语句系列(可能不止一个语句,而是若干个语句)。如不溢出,则不执行这个语句序列而执行下一个句子,见图5.1.

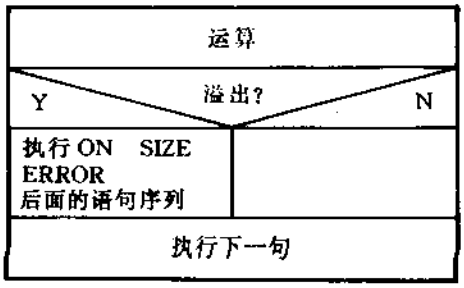


图 5.1

在程序设计中,最好使用 ON SIZE ERROR 子句.否则发生错误(如用零作除数,发生溢出)时都简单地按截去高位处理,计算机并不通知任何信息,使程序运行结果整个错了。

说明:(1) 当发生溢出时,在用 ON SIZE ERROR 子句时,错误的结果不存入结果数据项。如 C 的内容原为56。进行 A×B⇒C 的运算后,情况如下:

A 值	B 值	C 描 述	有 ON SIZE ERROR 子句否	C 结 果
1.2	9.0	9V9	无	08
1.2	9.0	9V9	有	56

即 A×B=10.8,在溢出时,C 值不改变,结果不存入。

(2) 如果有几个计算结果,有的发生溢出,有的未溢出,则未发生溢出的,计算结果正常地存入结果数值项,只有发生溢出的结果不存入,如 A=12,B=24,C=95,都用 PIC 99描述,执行:

```
ADD A TO B, C
ON SIZE ERROR GO TO A1.
```

计算结果 B=36,未溢出,C 应等于107,发生溢出,不存入,C 内容仍保持为95。但整个语句仍按溢出处理,转移到 A1段。

(3) 当 ROUNDED 与 ON SIZE ERROR 子句一起用时,先按 ROUNDED 作舍入处理,再判断是否溢出。例如:A 原值等于7.6,执行:

```
ADD 2.36 TO A ROUNDED ON SIZE ERROR STOP RUN.
```

2.36+7.6=9.96,若 A 的描述为9V9,当无 ROUNDED 子句时,将小数在9.9后截断,A 值变成9.9,无溢出。但在用了 ROUNDED 子句后,9.96四舍五入变成10.0,溢出,因此执行溢出处理,停止运行(因为 ON SIZE ERROR 子句中要求按"停止运行"处理)。

举一例说明其应用:

假如工人日岗位补贴不超过3元,每月总数不应超过100元。但如果由于输入数据有错误,算出月补贴>100元,不应按截去高位处理(如不应将算出的105元作为5元),而应按出错处理,通知程序员检查出错原因并纠正之。

用如下语句：

```
MULTIPLY PAY (月工资) BY DAYS(工作日)
        GIVING GROSS(总工资)
        ON SIZE ERROR
            DISPLAY"GROSS PAY EXCEEDS $99.99."
```

在数据部中 GROSS 用 PIC 99V99描述，最多容纳99.99元，超过则打印出一句事先安排好的、通知出错的信息。

读者可将其编写成一个完整的 COBOL 程序。

5.2.3 对应项间的运算(带 CORRESPONDING 子句的算术运算语句)

算术运算语句中的 ADD，SUBTRACT 语句除了可以用来使两个或多个单个的数据项进行加减运算外，还可以用来使两个组合项中的对应项进行加、减的计算。如有：

```
02  A.                      |  02  B.
  03 A1 PIC 9(3).           |    04 A1 PIC 99V99.
  03 A2 PIC 9(2)V99.        |    04 A3 PIC 9V9.
  03 A3 PIC 9V9.            |    04 A2 PIC 99V99.
```

可用对应项相加语句：

```
ADD CORR A TO B.
```

其中 CORR(或 CORRESPONDING)的含义与上一节中介绍的相同，即对应项之间的操作。上面的加法语句的作用是：将 A 中的 A1，A2，A3分别加到 B 中的 A1，A2，A3中去。它相当于：

```
ADD A1 OF A TO A1 OF B
ADD A2 OF A TO A2 OF B
ADD A3 OF A TO A3 OF B.
```

应注意，进行运算的各项必须是数值型初等项。如果 A1、A2、A3的描述是字符型. 如：

```
02  A.                      |  02  B.
  03 A1 PIC X(3).           |    04 A1 PIC 99V99.
  03 A2 PIC X(4).           |    04 A3 PIC X(3).
  03 A3 PIC 9V9.            |    04 A2 PIC X(4).
```

是不能用 CORR 进行对应项间运算的。

带 CORR 子句的运算语句是很有用的，特别在将多个数据记录中的数据项累加时用它就很方便。如将每个工人的工资数据(基本工资、附加工资、房租……)累加为全车间工人的基本工资和、附加工资和……。数据部中两个记录名不同，而它们下属的各相应的初等项名字相同，用一个带 CORR 子句的 ADD 语句，就可以将 A 记录中各有关数据累加到 B 记录中同名的对应项中。

一般格式：

```
ADD {CORRESPONDING / CORR} 标识符1 TO 标识符2 [ROUNDED]
    [;ON SIZE ERROR强制语句]
```

SUBTRACT { CORRESPONDING / CORR } 标识符1 FROM标识符2 [ROUNDED]
[;ON SIZE ERROR强制语句]

注意：只有加法(ADD)语句和减法(SUBTRACT)语句可以有 CORR 操作，乘法(MULTIPLY)语句、除法(DIVIDE)语句和计算(COMPUTE)语句是不能带 CORR 子句的。

5.2.4 除法语句中的余数子句(REMAINDER 子句)

以前介绍过的除法语句只能求商，不能求余数。如：

```
DIVIDE 1.5 INTO 7 GIVING C.
```

如果 C 用 PIC 9来描述，$\frac{7}{1.5}$的商的整数部分为4，余数为1，在 C 中存放数值4，余数无法表示。如果想求出余数，可用 REMAINDER 子句。如：

```
DIVIDE 1.5 INTO 7 GIVING C REMAINDER D.
```

商的整数部分放在 C 中(值为4)，余数放在 D 中(值为1)。

再举一例：如果有：

```
77 A PIC 9V9     VALUE 6.0.
77 B PIC 99V9    VALUE 16.3.
77 C PIC 9V99.
77 D PIC 9V99.
```

执行 DIVIDE A INTO B GIVING C REMAINDER D 后，结果怎样呢？$\frac{16.3}{6}$的值应为：2.71666……(见下式)：

```
      2.7166
   ------------
6 ) 16.3
    12
    ---
     43
     42
     --
      10
       6
      --
       40
       36
       --
        40
        36
        --
         4
```

今商放在 C 中，而 C 的描述为9V99，只能放两位小数，即2.71，余数是多少？$\frac{16.3}{6}=2.71$余0.04，因此，D 中值为0.04。

说明：(1) 商和余数的值不仅取决于被除数和除数，还取决于数据部中对商和余数的描述。如上面的 C，如描述为9V99，得商2.71，余数0.04。商如果 C 的描述变为9V9，则上面的除法运算只应进行到商为2.7，见下面除法计算。C 值为2.7，余数为0.1，即 D 值为0.1。

```
      2.7(商)
6 ) 1 6.3
    1 2
      4 3
      4 2
        1……(余数0.1)
```

如果C的描述改为9V99，而D的描述为V9，则C值为2.71，D值为0.0。

(2) 如用ROUNDED子句，它只对商起作用。余数不作四舍五入处理。在计算余数时，仍按四舍五入前的值为准。

如：DIVIDE A TO B GIVING C ROUNDED REMAINDER D.

如C的描述仍为9V99，当无ROUNDED子句时，C的值为2.71，今有ROUNDED子句，则应再看2.71后面一位数是什么，以便决定如何四舍五入。今知道再往下求一位得2.716，应四舍五入为2.72，但余数仍以除到2.71时为准，即0.04。

```
      2.7 1(6)……商
6 ) 1 6.3
    1 2
      4 3
      4 2
        1 0
          6
          4……(余数为0.04)
```

表5.2列出当C和D为不同描述以及有无ROUNDED子句的情况下执行下面语句后所得的结果。

DIVIDE A TO B GIVING C [ROUNDED] REMAINDER D.

表 5.2

B	A	C的描述	D的描述	有无ROUNDED	C 的 值	D 的 值
16.3	6.0	9V99	9V99	无	271	004
16.3	6.0	9V99	9V99	有	272	004
16.3	6.0	9V99	V9	无	271	00
16.3	6.0	9V9	9V9	无	27	01
16.3	6.0	9V9	9V9	有	27	01
16.3	6.0	9V999	9V999	有	2717	0004

(3) 长度溢出也只检查商的值是否溢出，而不检查余数。如商的值溢出，则执行ON SIZE ERROR后面的语句序列，不将求出的商值放入商区中，余数仍置于余数数据项中。

```
DIVIDE 8 INTO 1006 GIVING C REMAINDER D
        ON SIZE ERROR GO TO X.
```

如C的描述为99V9，D的描述为9V9，按C的描述，应除到小数点后一位，得125.7，余数为

0.4。但C只能放整数二位，因此发生溢出，故C内容不是1257，也不是257，它不置入C内，C保持原值。执行ON SIZE ERROR后面的语句——GO TO X，而D中放入余数值04。

§5.3 IF语句的较高技巧

我们已在第二章中学习过IF语句的最简单的形式。并已介绍了IF语句的一般格式：

```
IF条件 { 语句组1          } [ ;ELSE语句组2          ]
       { NEXT SENTENCE    } [ ;ELSE NEXT SENTENCE   ]
```

其中语句组1、语句组2可以是单个语句，也可以是一个语句组(或称语句序列。包括若干个语句)。还可以包括另一个IF语句，即IF语句可以嵌套。

5.3.1 IF语句的嵌套

先看下面的例子：

【例5.2】 如果已输入一个值Q。今要检查Q的值，如果1000<Q<5000，则显示Q的值，否则不显示。

可以写出下面的IF语句。

```
IF Q>1000  IF Q<5000 DISPLAY Q.
```

在IF Q >1000后面又嵌套了一个IF语句，即把"IF Q<5000 DISPLAY Q"看作一个语句。它符合：

IF 条件 语句

的形式。见图5.2。

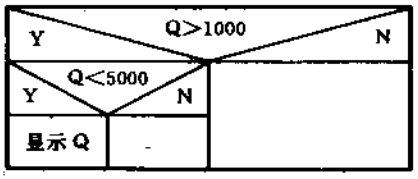


图 5.2

为了使层次关系清楚。在书写IF语句时，常按其嵌套关系分成几行，互相错开。

```
IF  Q>1000
  IF  Q<5000
    DISPLAY  Q.
```

【例5.3】 当金额AMOUNT≤50时，利率RATE为0.02；当50<AMOUNT<100时，RATE为0.03；当AMOUNT≥100时，RATE为0.04。

可写成IF语句：

```
IF  AMOUNT<100
```

```
    IF  AMOUNT>50
        MOVE 0.03 TO RATE
    ELSE MOVE 0.02 TO RATE
ELSE
        MOVE 0.04 TO RATE.
```

这样写层次分明，比较清晰，第一个 IF，先判断 AMOUNT＜100?，若不小于100，则执行最后一个 ELSE 后面的语句 MOVE 0.04 TO RATE，如 AMOUNT＜100则执行第二行到第四行这一个 IF 语句：

```
IF  AMOUNT>50 MOVE 0.03 TO RATE
ELSE MOVE 0.02 TO RATE
```

这是一个带 ELSE 的 IF 语句。

经过这样的分析，可以看到，整个 IF 语句的结构是比较清楚的。它实际上是以下形式的 IF 语句。

```
IF  条件  语句1
ELSE  语句2
```

其中“语句1”，又是一个带 ELSE 的 IF 语句。流程图见图5.3。

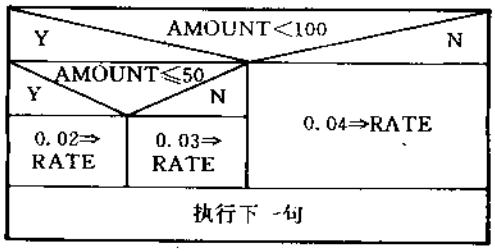


图 5.3

注意，为了使层次清楚，建议在写 IF 语句时分层写，一个 IF 对应一个 ELSE。内层的向内缩进几个空格，形成锯齿形状。见例5.4。

【例 5.4】 如果有以下一 IF 语句：

```
IF  A = B
    MOVE  B  TO  T
        IF  A = C
            MOVE  C  TO  R
                IF  X < Y
                  SUBTRACT  X  FROM  Y
                      IF  N = M
                        IF  P = Q
                          DISPLAY  P, Q.
```

没有一个 ELSE，如果当中某一个 IF 条件，(如 X＜Y)不成立时，应执行哪一个语句呢?一下子可能看不清楚，因此，当嵌套层次多时，最好加上 ELSE，与 IF 一一配对。

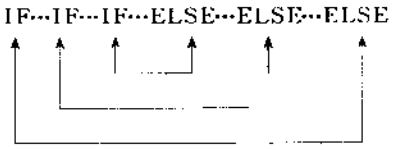
```
IF  A=B
    MOVE  B  TO  T
        IF  A=C
            MOVE  C  TO  R
                IF  X<Y
                    SUBTRACT  X  FROM  Y
                        IF  N=M
                            IF  P=Q
                                DISPLAY  P,Q
                            ELSE  NEXT  SENTENCE
                        ELSE  NEXT  SENTENCE
                ELSE  NEXT  SENTENCE
        ELSE  NEXT  SENTENCE
ELSE  NEXT  SENTENCE.
```

这样层次关系一目了然，每一层中的 IF 中的条件满足与否都可确无误地表示出其应执行的语句。

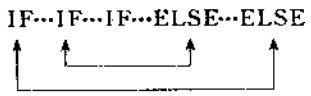
在看程序时，如果遇到 IF 与 ELSE 不配对，怎样分析其层次关系？请看例5.5。

【例 5.5】
```
IF   X = Y
     IF  A = B
         DISPLAY  'YES'
     ELSE  DISPLAY'NO'.
```

有两个 IF，只有一个 ELSE。这个 ELSE 应属于与内层的 IF 配对还是与外层的 IF 配对？很易搞乱，一般是这样分析其关系的：先找出最内层的 ELSE，找到与它最邻近的 IF 与之配对，然后逐层往外、一个 ELSE 与一个 IF 配对。即：



写 IF 语句时，最好使 IF 与 ELSE 的个数相同，以免混乱，如果 IF 与 ELSE 个数不同，则从最内层开始配对，认为外层多余的 IF 是无 ELSE 与之配对的：



因此，对例5.5，可以将其关系理解为：

```
IF  X=Y  语句1
```

语句1是一个带 ELSE 的 IF 语句。当 X≠Y 时，应执行整个 IF 句子的下一句。也可以写成：

```
IF  X = Y
    IF  A = B
        DISPLAY  'YES'
    ELSE  DISPLAY  'NO'
ELSE  NEXT  SENTENCE.
```

这样就清楚了。

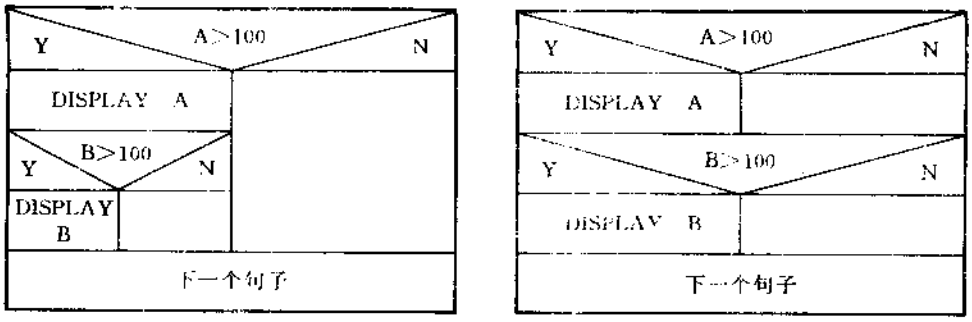
【例5.6】 在IF语句中，特别是在嵌套的选择结构中，句点的位置具有重要意义。在嵌套的选择结构中，只能在最外层的选择结构中使用一个句点。见下面两种写法：

```
(1)  IF  A > 100
         DISPLAY  A
         IF  B > 100
             DISPLAY  B.
```

流程图见图5.4(a)，第二个IF语句是内嵌的IF语句。当A≤100时，即使B>100，也不显示B。

```
(2)  IF  A > 100
         DISPLAY  A.
         IF  B > 100
             DISPLAY  B.
```

流程图见图5.4(b)。当A≤100时，若B>100，则显示B。



(a)　　(b)

图 5.4

5.3.2 关系表达式条件

迄今为止，我们在IF语句中所用的"条件"只是"关系表达式条件"。所谓关系表达式是以一个关系比较符将两个数据项联系起来的式子，如：

```
IF  X > Y  DISPLAY X.
```

其中，X和Y是两个需比较的数据项，">"(大于)是关系比较符，X>Y是关系表达式。当满足"X>Y"条件时，便认为此关系表达式的值为"真"(条件成立)，不满足"X>Y"条件，关系表达式的值为"假"(条件不成立)。

关系比较符有两种表示形式：一种是接近数学符号的，如<，>，=等。一种是接近英语

的，如：GREATER THAN，LESS THAN，EQUAL TO，NOT LESS THAN 等。我们认为：对非英语国家来说，最好尽量少用不必要的英文字，如：

X > 0和 X IS GREATER THAN ZERO

作用相同，但前者显然更直观些，读者更易理解其意思。

在关系符左边的项称为比较的“主体”(Subjcet)，关系符右边的项称为“客体”(Object)。主体和客体可以是数据名、常量或表达式，但主体和客体不能同时是常量。如 IF 3>2……是无意义的。

组合项可以参加比较，但它是作为字符型数据项来处理的，尽管此组合项内都由数值型初等项组成，也按字符一一比较。如：

```
01  A.                    01  B.
    02 A1 PIC 9(3).            02 B1 9(4).
    02 A2 PIC 9(2).            02 B2 9.
```

A				
A1			A2	
3	1	5	7	8

B				
B1				B2
2	8	7	6	4

如将 A 和 B 比较，A 并不作为三万一千五百七十八，B 不作为二万八千七百六十四。而是将 A 和 B 作为字符型数据，将他们相应的字符一一比较，如同一般的字符一样。

下面是关系条件的比较方式：

表 5.3

主体＼客体	数值型	数值常量	非数值常量	字母型	字符型	组合项
数值型	N	N	C	C	C	C
数值常量	N	×	×	C	C	C
非数值常量	C	×	×	C	C	C
字母型	C	C	C	C	C	C
字符型	C	C	C	C	C	C
组合项	C	C	C	C	C	C

其中：N：表示作为数值型比较

C：表示作为非数值型(即字符型)比较

×：表示不能比较

可以看出：主体或客体之一是非数值型的，则一律按非数值型比较，如：

```
IF  X>3 DISPLAY  X.
```

如 X 为字符型数据，内容为“T”，则把3也作为字符“3”，与“T”作字符比较。

关系表达式条件是 IF 语句中用得最多的一种条件形式，读者应熟练地掌握它。

5.3.3 符号条件

用来检查某数据项的值的代数符号。如：

IF X IS POSITIVE DISPLAY X.

其中"X IS POSITIVE"就是一个"符号条件"。显然它的形式与前面介绍的"关系表达式"条件不同，它不出现关系比较符(>，<，= 等)。它的意思是："如果 X 值的符号是正的，则显示 X 的值".

IF Y IS NEGATIVE MOVE Y TO X.

意思是："如果 Y 值的符号为负，将 Y 值送到 X 中"。

IF X+Y IS ZERO STOP RUN.

意思是："如果 X+Y 的值等于零。则停止运行"。

IF 后面可以是一个数值型数据名(如上面的 X，Y)，也可以是一个算术表达式(如 X+Y)。不能是字母型或字符型数据，因为它们不可能有代数符号。

(符号条件的一般形式：)

$$\left\{\begin{matrix}\text{数据名}\\ \text{算术表达式}\end{matrix}\right\} \text{ IS } [\underline{\text{NOT}}] \left\{\begin{matrix}\underline{\text{POSITIVE}}\\ \underline{\text{NEGATIVE}}\\ \underline{\text{ZERO}}\end{matrix}\right\}$$

符号条件用来作定性的测定，如商品的库存量，不应为负，如负就是出错。银行业务中，出现负表示透支。商品数等于零表示脱销等。

以下符号条件与关系表达式条件等价：

如：(1) IF X IS POSITIVE

与 IF X>0 等价

(2) IF X NEGATIVE

与 IF X<0 等价

(3) IF X IS ZERO

与 IF X=0 等价

因此，一般用关系表达式条件来代替符号条件，用起来比较方便，看起来比较清晰。

5.3.4 类型条件

用来检查数据项的类型是否符合指定的要求。有时要求检查数据项中的内容是否全为数字，或全为字母. COBOL 提供检查数据类型的功能。

(一) 检查是否数字数据

IF X IS NUMERIC MOVE B TO C.

意思是：如果 X 的值是纯数字的数据，则将 B 的值传送给 C(允许 X 的值出现正、负号)。

如果 X 的描述为999或 X(3)，内容为123，则执行此语句后，都使 B⇒C。但如 X 的描述为 A(3)，则不能用 NUMERIC 来作检查。因为字母型数据项是不能存放数字的。即：类型条件的检查必须和被检查的项的数据描述一致。字符型数据项的内容可能是数字，也可能是字

母或其它字符。因此可以用 NUMERIC 来检查。

（二）检查是否字母数据

检查数据项中是否全是字母。如：

```
77  T  PIC  A(4)  VALUE 'WANG'.
77  Q  PIC  X(5)  VALUE 'W2D45'.
```

可以用：

```
IF  T  IS  ALPHABETIC DISPLAY T.
IF  Q  IS  NOT  ALPHABETIC STOP RUN.
```

前者表示：当 T 的内容全是字母时，显示 T。后者表示：当 Q 的内容不全是字母时，停止运行。执行第一个 IF 语句时，检查出 T 的内容全为字母，因此执行“DISPLAY T ”，接着执行第二个 IF 语句，Q 的内容不全为字母，因此满足条件，停止运行。

同样，用 ALPHABETIC 或 NOT ALPHABETIC 只能检查用“X”或“A”描述符描述的数据项的内容，不能用来检查以“9”描述的数据项的内容。如：

```
77  K  PIC  999.
 ⋮
```

不可用：

```
IF  K  ALPHABETIC  GO  TO  C.
```

下表给出了各种类型的数据项可以使用的条件类型

标识符类型	可用的条件类型	
	肯定型	否定型
数值型	NUMERIC	NOT NUMERIC
字母型	ALPHABETIC	NOT ALPHABETIC
字符型	NUMERIC ALPHABETIC	NOT NUMERIC NOT ALPHABETIC

类型条件的一般格式为：

```
标识符  IS  [NOT]  { NUMERIC
                   { ALPHABETIC
```

使用类型条件时，特别要注意上述的使用范围的规定。不选“NOT”时，用来检查数据中内容与要求的符合程度，选用“NOT”时，检查其不符合的程度。

譬如：A 是用99V99来描述的，不能用下面语句来检查它的内存中存放的是否为字母。

```
IF  A  IS  ALPHABETIC……
```

而应用：

```
IF  A  IS  NUMERIC……
```

或：IF A IS NOT NUMERIC……

来检查 A 中是否全为数字或不全为数字。

同样，如 X 用 A(4)来描述，不能用下面语句来检查它的内容是否为数字。

```
IF  X  NUMERIC…
```

而要用：

```
IF  X  NOT  ALPHABETIC…
```

或 `IF  X  ALPHABETIC…`

来检查 X 中是否全为字母或不全为字母。

如果字符型数据 N 的内容既有字母又有数字，为检查输入数据时有否发生错误，可以用：

```
IF  N  IS  NOT  NUMERIC
IF  N  IS  NOT  ALPHABETIC
    DISPLAY N.
```

判断 N 是否“不全是数字，又不全是字母”。

类型条件可以用来检查数据在输入时有否发生类型错误(即本来应输入字母型数据错输入为数值型，或反之)。如有一地址记录，在数据部中描述为：

```
FD  ADDRESS-FILE  LABEL  RECORD  IS  STANDARD.
01  ADDRESS-RECORD.
  02  NUMB PIC X(6).
  02  ADDRESS PIC X(30).
  02  TELEPHONE PIC X(6).
```

输入数据的形式为：

```
128375BEIJING  ROAD  NO.145      282456
  6              30                6
```

读入一记录后，先检查一下输入的数据类型有否错误。可用：

```
    ⋮
A. IF  NUMB  IS  NOT  NUMBERIC  GO  TO  A1.
   IF  ADDRESS  IS  NOT  NUMBERIC
       IF  ADDRESS  IS  NOT  ALPHABETIC
           NEXT  SENTENCE
       ELSE  GO  TO  A1
   ELSE  GO  TO  A1
   IF  TELEPHONE  IS  NOT  NUMERIC  GO  TO  A1.
B. (对记录处理)
    ⋮
A1.(出错处理)
```

如 NUMB 不全为数字，ADDRESSB 全是数字或全是字母，或 TELEPHONE 不全为数字，则转去 A1段作错误处理，否则执行 B 段，处理刚才读入的记录，流程图见图5.5。

5.3.5 条件名条件

除了用以上几种条件外，在 IF 语句中还可以使用“条件名条件”。

(一) 什么叫条件名

程序设计中，常常需要根据某一个数据项的值在不同的范围，决定不同的处理方法。

如：为鼓励存款，对存款数少于100元的，利息为3%，等于或大于100元而少于1000元的，

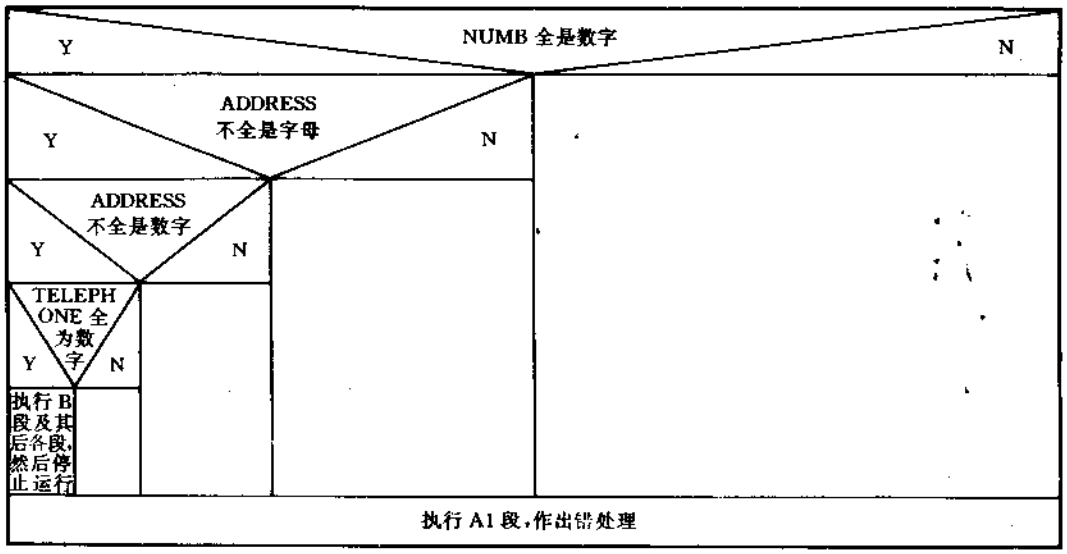


图 5.5

利息为4%，等于或大于1000元而少于5000元的，利息为5%，5000元以上的，利息为6%。见图5.6。

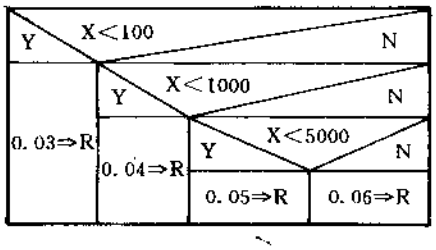


图 5.6

当然可以用过去学过的办法写出下面的 IF 语句。

```
IF  X  <  100
  MOVE  0.03  TO  R
ELSE
    IF  X  <  1000
      MOVE  0.04  TO  R
    ELSE
        IF  X  <  5000
          MOVE  0.05  TO  R
        ELSE  MOVE  0.06  TO  R.
```

如果划分的区间更小一些，则 IF 语句就更复杂，既麻烦又不直观。我们可以将 X 分成若干个区间，相应于不同的条件：

（条件变量）	X			
（条件）	0≤X<100	100≤X<1000	1000≤X<5000	5000≤X<100000
（条件名）	X1	X2	X3	X4

在上表中，X 的变化范围假设从0到100000元，按题意分为四个区间。我们分别用四个名字 X1，X2，X3和 X4来代表不同的条件。即凡满足0≤X<100条件时，以 X1来代表。满足100≤X<1000条件时，以 X2为代表，余类推。或者说，当满足0≤X<100时，X1为“真”，满足100≤X<1000时，X2为“真”，……。简单地、不严格地说，条件名是以一个名字来代表一个条件。以后用到 X1，就代表“0≤X<100”，用到 X2就代表“100≤X<1000”……等等。

一个初等项只能根据某些条件取预定的几个值，或只能在一个预定的范围中取值，则这个变量称条件变量。即它的取值是有条件的。如上例中 X 只能从0～100000间取值。X 称条件变量。

用来表示条件变量当前值的名字叫条件名。或者说，条件名本身是一种条件，它有一个值（“真”或“假”），用它来检查条件变量的值是否落在条件名所代表的值的范围中，当条件变量的值落在条件名所确定的值的范围中，则此条件名所表示的条件为“真”，否则为“假”。

（二）条件名如何说明

条件名和条件变量要建立一定的联系，这应在数据部中说明。先看具体的写法：

```
77  X  (条件变量)  PIC  9(6).
88  X1  VALUE  0      THRU      99.
88  X2  VALUE  100    THRU      999.
88  X3  VALUE  1000   THRU      4999.
88  X4  VALUE  5000   THRU      100000.
```

其中 X1，X2，X3，X4为条件名。条件名用层号88，紧跟在条件变量之后说明。上面四个88层的条件名紧跟在77层条件变量的说明之后，表示它们是条件变量 X 的具体说明，它们从属于条件变量，即 X1，X2，X3，X4是属于 X 范围之内的。

注意，对“88 X1 VALUE 0 THRU 99.”不要理解为“X1的值由0到99”，而应该理解为：X 的值由0到99范围时，X1为“真”。即以名字 X1代替0≤X≤99这一条件。

```
77    X     PIC    9(6).        (X 变化范围)
        |_____________|
88    X1    VALUE  0  THRU  99.
```

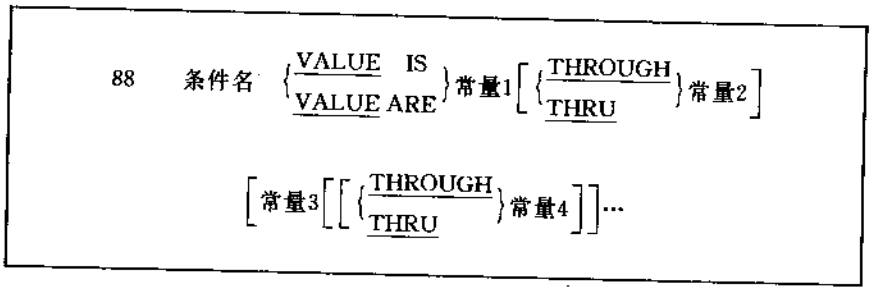
这点务必注意。

经过以上在数据部说明条件名的含义后，就可在过程部中直接使用条件名条件。

```
IF  X1  MOVE  0.03  TO  R.   (在0≤X<100时，R=0.03)
IF  X2  MOVE  0.04  TO  R.   (100≤X<1000时，R=0.04)
IF  X3  MOVE  0.05  TO  R.   (1000≤X<5000时，R=0.05)
IF  X4  MOVE  0.06  TO  R.   (5000≤X≤100000时，R=0.06)
```

显然这种写法比开始时用一长串的 IF—ELSE 语句来表达，要方便得多。它使过程部清晰、简单，用一个名字就代表了一个条件。

条件名条件的一般格式：



（三）条件名条件应用举例

【例 5.7】 假如人事部门统计职工受教育程度。凡上学年数<12年的，表示高中以下。等于12年的为高中毕业。13～15年的为大学肄业。16年的为大学毕业。16～20年的为研究生。在职工表格上填写上学年数，要求分别统计出每一种受教育程度的人数（高中以下、高中毕业、大学肄业、大学毕业、研究生）。每个职工的数据已存放在磁盘数据文件中 WORKER·DAT 中。

```
IDENTIFICATION      DIVISION.                    (标识部)
PROGRAM-ID. EXAM 57.
ENVIRONMENT         DIVISION.                    (环境部)
INPUT-OUTPUT    SECTION.
FILE-CONTROL.
      SELECT  WORKER-FILE          ASSIGN TO WORKER.
DATA      DIVISION.                              (数据部)
FILE    SECTION.
FD    WORKER-FI LE
                LABEL  RECORD IS STANDARD.
01 PERSONNEL-DATA.                               (员工数据)
  02  NUMB     PIC 9(6)                          (编号)
  02  NAME     PIC X(10)                         (姓名)
  02  ADDR     PIC X(10)                         (地址)
  02  EDUCATION   PIC 99.                        (受教育年数)
      88  LESS-THAN-H-S-GRAD  VALUE  0  THRU  11.   (高中毕业以下)
      88  H-S-GRAD            VALUE  12.            (高中毕业)
      88  SOME-COLLEGE        VALUE  13 THRU  15.   (大学肄业)
      88  COLLEGE-GRAD        VALUE  16.            (大学毕业)
      88  POST-GRAD           VALUE  17 THRU  20.   (研究生)
WORKING-STORAGE  SECTION.
01  TOTAL.
    02  FELLER    PIC  X(9)  VALUE  'TOTAL-1='.
    02  TOTAL-1   PIC  999   VALUE  0.
    02  FELLER    PIC  X(9)  VALUE  'TOTAL-2='.
    02  TOTAL-2   PIC  999   VALUE  0.
    02  FELLER    PIC  X(9)  VALUE  'TOTAL-3='
    02  TOTAL-3   PIC  999   VALUE  0.
    02  FELLER    PIC  X(9)  VALUE  'TOTAL-4='.
    02  TOTAL-4   PIC  999   VALUE  0.
    02  FELLER    PIC  X(9)  VALUE  'TOTAL-5='.
```

```
    02  TOTAL-5   PIC  999   VALUE  0.
PROCEDURE        DIVISION.  (过程部)
ST.     OPEN  INPUT       WORKER-FILE.
RE. READ  WORKER-FILE
              AT END GO TO WR.
CA.     IF  LESS-THAN-H-S-GRAD  ADD 1 TO TOTAL-1.
        IF   H-S-GRAD           ADD 1 TO TOTAL-2.
        IF   SOME-COLLEGE       ADD 1 TO TOTAL-3.
        IF   COLLEGE-GRAD       ADD 1 TO TOTAL-4.
        IF   POST-GRAD          ADD 1 TO TOTAL-5.
        GO TO RE.
WR. DISPLAY TOTAL.
        CLOSE  WORKER-FILE.
        STOP RUN.
```

条件名是在数据部中定义的。我们用了有意义的英文字来作条件名，以使看程序时容易懂其意思，不必另外多作说明。当然也可以用简单的符号(如 X1,X2…等)作条件名。在这程序中只用一个文件(输入文件)，不用输出文件，直接用 DISPLAY 语句输出。当然，读者也可以把它改写为用 WRITE 语句输出。

CA 段中的五个 IF 语句相当于以下五个 IF 语句：

```
IF  EDUCATION  LESS  THAN  12
            ADD  1  TO  TOTAL-1.
IF  EDUCATION  IS  EQUAL  TO  12
            ADD  1  TO  TOTAL 2.
IF  EDUCATION  IS  LESS  THAN  16
    IF  EDUCATION  IS  GREATER  THAN  12
          ADD  1  TO  TOTAL 3.
IF  EDUCATION  IS  EQUAL  TO  16
            ADD  1  TO  TOTAL-4.
IF  EDUCATION  IS  LESS  THAN  20
    IF  EDUCATION  IS  GREATER  THAN  17
         ADD  1  TO  TOTAL-5.
```

可以看到，这比用条件名条件麻烦多了。

(四) 说明：

(1) 88层条件名描述，是用于数据部中的，可用在工作单元节，也可用在文件节中。

(2) 文件节规定不允许用 VALUE 子句赋初值，而在88层条件名描述体中可以用 VALUE 子句。这是由于在88层中，VALUE 的作用不是赋初值，而只是用来给条件变量的一个可能值规定一个条件名。

5.3.6 复合条件

由若干个简单的“条件”可以组合成复合的条件。如：要求显示满足0<X<100条件的 X 的值，如果在 IF 语句中只用简单的条件，可以写为：

```
(1)     IF     X>0
               IF     X<100
```

```
                DISPLAY  X.
```

或 (2) IF X IS NOT>0 GO TO B.

```
    IF  X<100   DISPLAY   X.
    B.  ……
```

也可以将两个简单的条件(X>0和X<100),组合成一个条件,即复合条件。如:

```
IF   X > 0  AND  X < 100  DISPLAY  X.
```

表示同时满足X>0和X < 100两个条件。其中AND是"与"的意思,表示要求由它联接的两个条件都应满足。这个"AND"就是一种"逻辑运算符"。

COBOL用到的逻辑运算符有:AND(与)、OR(或)、NOT(非)。我们在图5.7中形象地表示它们的意思。

(1) A AND B表示A条件与B条件都满足,在图5.7(1)中可以看到,如果在左边圆中所有的点满足A条件,右边圆中所有的点满足B条件。则两个圆重叠的部分(有斜线的部分)同时满足A条件和B条件。

(2) A OR B,表示只要满足A条件或B条件之一即可。见图5.7(2)。两个圆中任何一点都是符合条件的(或者能满足A条件,或者满足B条件)。比方有10个人有红钢笔,有10个人有蓝钢笔,显然,如果问"多少人有钢笔",应该是20人(假如二者不重叠)。

(3) NOT A,表示不满足A条件。图5.7(3)中,假定圆内点满足A条件,则圆外点不满足A条件。比方一个班30人中,26个人考试合格。则不合格的是(30−26)=4人,即除26人外都不合格。

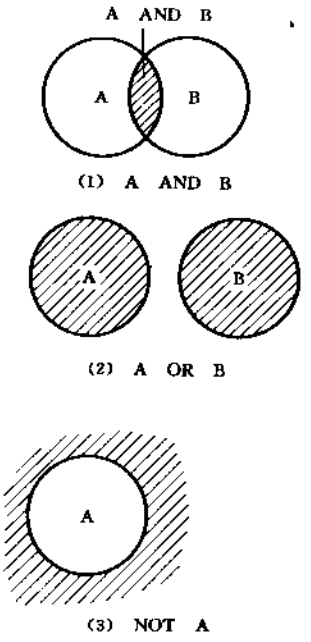


图 5.7

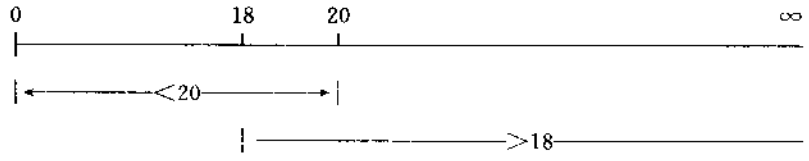
举几个例子:

```
AGE < 20   AND  AGE > 18
```

表示挑选年龄在18岁以上,20岁以下的对象。

```
AGE<20   OR   AGE>18
```

即凡小于20岁的和大于18岁的都满足条件。显然这个条件包括了一切人,因从下图可以看到任何年龄不是满足 AGE<20 的就满足 AGE>18。



如果在同一个IF语句中用到AND,OR NOT,运算的次序如何?按①NOT;②AND;③OR 的次序。即求复合条件的值的次序为:

(1) 先求出每一个简单条件的值(注意,所谓"条件的值",不是一个数值而是一个逻辑

值,即“真”(成立)或“假”(不成立))。

(2) 进行逻辑运算,先进行 NOT 运算。例如 NOT A,若 A 的值为“真”,则 NOT A 的值为假。

(3) 进行 AND 的运算。如果有几个 AND,按自左而右次序。

(4) 进行 OR 的运算。

如果有下面一个复合条件。其中 X＝3.5,Y＝4,T＝3,W＝2,C＝5,D＝4,G＝8。

```
IF  X = Y     OR  T = W  AND  C > D    AND   G IS POSITIVE
    ①假            ①假         ①真             ①真
                       ②假
                                   ③假
                     ④假
```

次序是:① 先算出关系表达式条件 X＝Y,T＝W,C>D 和符号条件 G IS POSITIVE 几个简单条件的值。② 进行第一个 AND 运算,由于“假”·“真”,因此结果为“假”。③ 进行第二个 AND 运算,结果为“假”。④ 进行 OR 运算,结果为“假”。

下面是逻辑运算的真值表:

A	B	A AND B	A OR B	A AND NOT B	A OR NOT B
真	真	真	真	假	真
假	真	假	真	假	假
真	假	假	真	真	真
假	假	假	假	假	真

§5.4 程序举例

【例 5.8】 统计学生中男女生的总数。

```
IDENTIFICATION      DIVISION.           (标识部)
PROGRAM-ID.   EXAM 58.
ENVIRONMENT         DIVISION.           (环境部)
INPUT OUTPUT   SECTION.
FILE-CONTROL.
       SELECT   STUDENT-FILE   ASSIGN TO STDT.
DATA            DIVISION.               (数据部)
FILE     SECTION.
FD    STUDENT-FILE   LABEL   RECORD   IS   STANDARD.
01   GZQD-R.
     02   STUDENT-NUM        PIC 9(6).        (学生号)
     02   STUDENT-NAME       PIC X(10).       (学生名)
     02   SEX-CODE           PIC X.           (性别码)
     02   AGE                PIC 999.         (年龄)
     02   CLASS              PIC 9.           (班级)
```

```
WORKING-STORAGE    SECTION.
77 FEMALE-COUNTER  PIC  999.              (女生累计)
77 MALE-COUNTER    PIC 999.               (男生累计)
PROCEDURE          DIVISION.              (过程部)
INITIALIZE.                               ("开始"段)
    OPEN  INPUT      STUDENT-FILE
    MOVE  ZERO   TO FEMALE-COUNTER.       (女生累计数)
    MOVE  ZERO   TO MALE COUNTER.         (男生累计数)
READ-FILE.                                ("读文件"段)
    READ  STUDENT-FILE
          AT END GO TO DISPLAY-ANSWER.
COUNTING.                                 ("统计"段)
    IF  SEX-CODE = 'F'
        ADD  1  TO  FEMALE-COUNTER
        GO  TO  READ-FILE.
    IF  SEX CODE = 'M'
        ADD  1  TO  MALE-COUNTER
        GO  TO  READ-FILE.
    DISPLAY 'ERROR IN SEX CODE ', SEX-CODE','␣␣␣',STUDENT-NUM
    GO TO READ-FILE.
DISPLAY-ANSWER.    ("显示结果"段)
    DISPLAY 'FEMALE = ', FEMALE-COUNTER.
    DISPLAY 'MALE = ',MALE-COUNTER.
    CLOSE  STUDENT-FILE.
    STOP RUN.
```

程序流程比较简单,读者不难自己看懂。

程序中的 COUNTING 段也可以改写如下:

```
COUNTING.
    IF  SEX-CODE = 'F'
        ADD  1  TO  FEMALE-COUNTER
        GO  TO  READ-FILE
    ELSE
        IF  SEX-CODE='M'
            ADD  1  TO  MALE-COUNTER
            GO  TO  READ-FILE
        ELSE
            DISPLAY  'ERROR  IN  SEX  CODE',
              SEX-CODE,'␣␣␣',STUDENT-NUM.
        GO  TO  READ-FILE.
```

【例 5.9】 从库存零件文件读入记录,然后输出库存清单。

文件中的数据为

1～4列　　零件号

5～8列　　库存量

9～12列　　单　价

要求输出的格式为：

```
                    INVENTORY  REPORT (库存报表)

PART NUM QUANTITY ON HAND PRICE EACH              AMOUNT
    (零件号)        (库存量)          (单价)        (价值)

      ××××          ××××          $××.××    $××××××.××
      ××××          ××××          $××.××    $××××××.××
        ⋮             ⋮             ⋮           ⋮
                  END  OF  REPORT(报表结束)
```

```
IDENTIFICATION      DIVISION.          (标识部)
PROGRAM-ID.         EXAM 59
ENVIRONMENT         DIVISION.          (环境部)
INPUT OUTPUT  SECTION.
FILE CONTROL.
       SELECT  INVENTORY FILE  ASSIGN  TO  INV-DATA.
       SELECT  REPORT-FILE  ASSIGN  TO  P FILE.
DATA        DIVISION.                  (数据部)
FILE    SECTION.
FD    INVENTORY-FILE  LABEL  RECORD IS  STANDARD.     (库存文件)
01  PART-RECORD.                       (零件记录)
    02 PART-NUMBER  PIC  9(4).              (零件号)
    02 QTY-ON HAND  PIC  9(4).              (库存数)
    02 PRICE-EACH     PIC  99V99.           (单价)
FD    REPORT-FILE  LABEL  RECORD  IS STANDARD.     (报表文件)
01  REP-LINE.                          (报表行)
    02 FILLER            PIC X.
    02 REP-REC           PIC X(80).
WORKING-STORAGE   SECTION.
01  PAGE-HEADING LINE.                 (页标题行)
    02 FILLER  PIC X(20) VALUE SPACE.
    02 FILLER  PIC X(16) VALUE 'INVENTORY REPORT'.
01  COLUMN-HEADING-LINE.               (列的标题行)
    02 FILLER       PIC X.
    02 FILLER       PIC X(8) VALUE 'PART NUM'.
    02 FILLER       PIC X(5) VALUE  SPACES.
    02 FILLER       PIC X(16) VALUE'QUANTITY ON HAND'.
    02 FILLER       PIC X(5) VALUE  SPACES.
    02 FILLER       PIC X(10) VALUE'PRICE EACH'.
    02 FILLER       PIC X(5) VALUE  SPACES.
    02 FILLER       PIC X(6) VALUE'AMOUNT'.
01  END-OF-JOB-LINE.                   (报表结束行)
    02 FILLER       PIC X(23) VALUE SPACES.
    02 FILLER       PIC X(16) VALUE 'END OF REPORT '.
```

```
01  DETAIL LINE.                                    (细目行)
    02 FILLER        PIC X(3) VALUE SPACES.
    02 PART-NUMBER   PIC 9(4).                      (零件号)
    02 FILLER        PIC X(13) VALUE SPACES.
    02 QTY-ON-HAND PIC Z(4).                        (库存数)
    02 FILLER        PIC X(13) VALUE SPACES.
    02 PRICE-EACH PIC   $ $.$ .99.                  (单价)
    02 FILLER        PIC X(4) VALUE SPACES.
    02 INVENTORY-VALUE   PIC $(7).99.   (库存值)
PROCEDURE              DIVISION.                    (过程部)
OPEN -LINE.          (打开文件)
       OPEN   INPUT        INVENTORY-FILE
              OUTPUT       REPORT-FILE.
      MOVE   SPACE   TO   REP-LINE.
O HEADING LINES.                                    (输出表头)
    WRITE   REP-LINE   FROM   PAGE-HEADING-LINE
            AFTER   1.        (输出页头)
    WRITE   REP LINE   FROM   COLUMN-HEADING-LINE
            AFTER   2.        (输出列标题)
    MOVE   SPACES   TO   REP-LINE.
    WRITE   REP-LINE   AFTER   1.          (空一行)
READ-RECORD.                                        (读记录)
    READ   INVENTORY FILE
           AT END GO TO JOB END.        (读库存文件一个记录)
    MULTIPLY   QTY-ON HAND   OF   PART RECORD
               BY   PRICE-EACH   OF   PART-RECORD
               GIVING   INVENTORY VALUE.        (计算库存值)
    MOVE   CORR   PART-RECORD   TO   DETAIL LINE.     (对应传送)
    WRITE   REP-LINE   FROM   DETAIL-LINE   AFTER 1.     (输出一个细目行)
    GO   TO   READ RECORD.
JOB END.            (作为结束)
    WRITE   REP-LINE FROM END OF-JOB-LINE AFTER   2.
    CLOSE   INVENTORY-FILE REPORT-FILE.
    STOP   RUN.
```

假设库存文件中有以下记录：

```
000101002323
000204344344
000302422224
000493444939
000538383838
```

输出报表格式如下：

INVENTORY REPORT

PART NUM	QUANTITY ON HAND	PRICE EACH	AMOUNT
0001	100	$ 23.23	$ 2323.00
0002	434	$ 43.44	$ 18852.96
0003	242	$ 22.24	$ 5382.08

```
0004                    9344        $49.39          $461500.16
0005                    3838        $38.38          $147302.44

                        END OF REPORT
```

这个程序是比较好懂的，请读者注意：(1)是如何打印“表头”的，这个技巧以后是有用的。(2) 对应项的传送语句包括哪些具体的操作。

【例 5.10】 人事部门要了解职工中对工作安排的志愿和其它情况。每个职工的记录中包括：姓名、性别(以1代表男，2代表女)、受教育年数、实践工作年数(工龄)、政治面貌(以1代表党员、2代表团员、3代表群众)、工作地区志愿(以1代表东北、2代表西北、3代表南方、4代表西南、5代表其它)。要求将符合以下条件的人的全部数据打印出来：大学毕业、党员、男、二年工龄、志愿去西北者。

```
IDENTIFICATION          DIVISION.          (标识部)
PROGRAM-ID.             EXAM 510.
ENVIRONMENT             DIVISION.          (环境部)
INPUT OUTPUT  SECTION.
FILE CONTROL.
      SELECT  INPUT-FILE    ASSIGN  TO  PER-DATA.
      SELECT  OUTPUT FILE   ASSIGN  TO  P FILE.
DATA          DIVISION.                  (数据部)
FILE  SECTION.
FD    INPUT-FILE LABEL  RECORD IS STANDARD.
01  PERSONNEL RECORD.
    02 NAME       PIC X(10).             (姓名)
    02 FILLER  PIC  X(3).
    02 PARTY      PIC 9.                 (党派)
       88 CP     VALUE 1.       (党员)
       88 CY     VALUE 2.       (团员)
       88 MASS VALUE 3.        (群众)
    02 FILLER  PIC  X(3).
    02 SEX  PIC 9.
       88  MALE    VALUE  1.       (男性)
       88  FEMALE  VALUE  2.       (女性)
    02 FILLER  PIC X(3).
    02 EDUCATION  PIC 99.
       88 LESS THAN H S GRAD  VALUE  0  THRU  11.    (高中毕业以下)
       88 H-S-GRAD           VALUE  12.              (高中毕业)
       88 SOME-COLLEGE  VALUE  13  THRU  15.  (大学肄业)
       88 COLLEGE-GRAD  VALUE  16.            (大学毕业)
       88 MASTERS-GRAD  VALUE  18  THRU  20.  (硕士学位)
    02 FILLER  PIC X(3).
    02 YEAR-OF-EXPERIENCE  PIC  99.    (工作实践年数)
    02 FILLER  PIC X(3).
    02 GEOGRAPHIC-PREFERENCE  PIC 9.    (地区志愿)
       88 NORTH-EAST     VALUE  1.       (东北)
       88 NORTH-WEST    VALUE  2.       (西北)
       88 SOUTH          VALUE  3.       (南方)
       88 SOUTH-WEST    VALUE  4.       (西南)
```

```
        88 OTHER              VALUE  5.          (其它)
FD     OUTPUT-FILE LABEL  RECORD  IS STANDARD.
01  OUTPUT-RECORD.
    02  FILLER              PIC X.
    02  OUT LINE            PIC X(80).
PROCEDURE          DIVISION.        (过程部)
START.
    OPEN  INPUT        INPUT FILE
          OUTPUT       OUTPUT-FILE.
    MOVE  SPACE  TO  OUT LINE.
READ-RECORD.
    READ  INPUT-FILE
          AT END CLOSE  INPUT FILE, OUTPUT FILE
                 STOP RUN.
PROC.
    IF CP AND MALE AND YEAR-OF-EXPERIENCE = 02
      AND COLLEGE GRAD AND NORTH-WEST
      MOVE PERSONNEL RECORD  TO OUT LINE
      WRITE  OUTPUT-RECORD  AFTER  2.
      GO TO READ-RECORD.
```

本例输出的结果只是最简单的情况，如果需要输出报表头，读者可以自己补充修改此程序，并不难。

从此例，可以看到使用复合条件，可以避免使用 IF 语句的嵌套，使程序简明。

* §5.5 字符串连接语句(STRING 语句)

除了计算功能以外，COBOL 提供了字符串处理的功能，以下几节都属此。

(一) STRING 用来将多个非数值型的数据项的值连接起来送到一个接收数据项中。即将多个字符串连接成一个字符串。在合并过程中可以删除某些指定的字符。

例如，已指定数据项 A，B，C 均为非数值型数据项，以 PIC X(4)描述，D 用 PIC X(16)描述。

```
STRING  A, B, C  DELIMITED BY  SIZE  INTO  D.
```

从 A 的最左边字符开始由左而右地将 A 的内容向 D 传送(从 D 的最左位置开始接收)。有一个逻辑指针，它指出接收项当前接收字符的位置，其初始值为1(从接收项左边第一个字符开始接收)，每接收一个字符指针值就加1。把 A 的内容送完后，接着送 B 的内容，然后送 C 的内容。如果数据项 A，B，C 的内容分别为"FGH ␣"、"KLM ␣"，"XYZ ␣"，则 D 的内容为：

```
FGH ␣ KLM ␣ XYZ ␣␣␣␣␣
```

A，B，C 三个数据项共有12个字符，而 D 为16个字符长，未被送入字符的位置(最后四个字符位置)保留原内容不变。今假设 D 原内容为空白，故最后四个字符仍为空白。

WDELIMITED BY SIZE 的意思是将按发送项的长度全部传送到接收项。

(二) DELIMITED 短语(定界短语)。用来控制各个发送项的终止位置。如果写上 DELIMITED BY SIZE，表示将整个发送项的字符全部送到接收项，如上面所示。有时我们只想传送一部分字符而删去另一部分，譬如，希望数据项 A，B，C 中的空格(最右边一个字符)不

传送到 D，则可写：

```
STRING A, B, C DELIMITED BY SPACE INTO D.
```

意即传送时遇 SPACE(空格)即截止该数据项的传送。此时 D 的内容为：

FGHKLMXYZ␣␣␣␣␣␣␣

也可以用其它的字符作定界符。如果各个发送项所需的定界符不同，可以分开对每一发送项单独使用 DELIMITED 短语。如：

```
STRING  A  DELIMITED  BY  'H'
        B  DELIMITED  BY  'M'
        C  DELIMITED  BY  'Y'
        INTO  D.
```

则 D 的内容为：

FGKLX␣␣␣␣␣␣␣␣␣␣␣

可以在传送中插入所需字符，如：

```
STRING A,'␣', B , '␣',C DELIMITED BY SIZE INTO D.
```

在 A，B 之后各插入一个空格，D 的内容为：

FGH␣␣KLM␣␣XYZ␣␣␣

定界符不仅可以是一个字符，也可以是字符串，如 DELIMITED BY 'XY'。也可以是一个非数值型的数据项，以其值定界。

(三) 如果不想从接收项的最左端开始接收字符而从中间某一字符位开始向右接收字符，可以用 POINTER 短语(指针短语)如：

```
MOVE  3  TO  T.
STRING  A, B, C  DELIMITED  BY  SIZE
        WITH  POINTER  T  INTO  D.
```

如果 T 的值为3，表示从 D 的左边第三个字符开始接收，接收后 D 的内容为：

␣␣EFG␣KLM␣XYZ␣␣␣

POINTER 就是上面说的逻辑指针，今使其初值为3。

(四) 如果逻辑指针值小于1或已大于接收项包含的字符个数，不能正常执行字符传送，发生"溢出"。例如 D 的长度改为10个字节，则执行

```
STRING A,B,C DELIMITED BY SIZE INTO D.
```

时放不下十二个字符，在放完十个字符后不再执行 STRING 语句，D 的内容为：

FEG␣KLM␣XY

接着执行下一个语句。也可以用 OVERFLOW 短语(溢出短语)来判断是否发生溢出，以及指定当发生溢出时所作的处理。如：

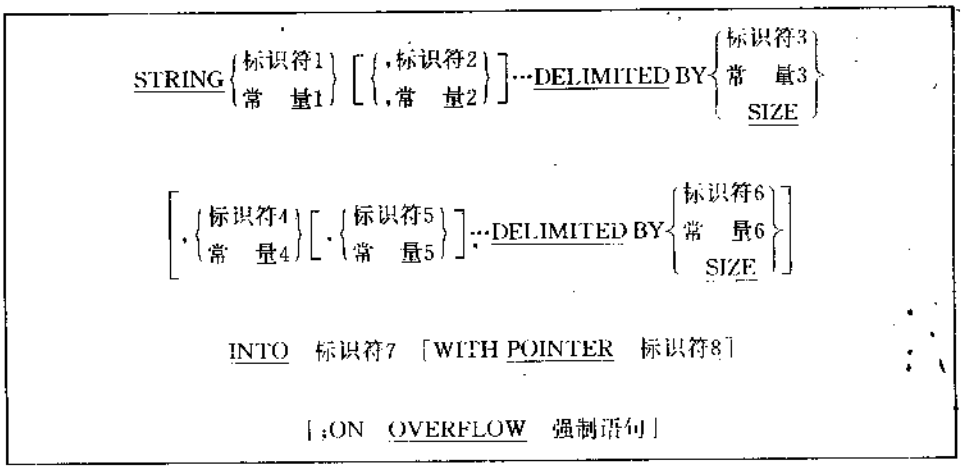
```
STRING  A,B,C  DELIMITED  BY  SIZE  INTO  D
        ON  OVERFLOW  DISPLAY 'OVERFLOW'.
```

在发生溢出时显示出"OVERFLOW"信息。

(五) STRING 语句的一般格式见下页图所示。

(六) 说明：

(1) 接收字符串的数据项(标识符7)必须是初等项，且不能有 JUST RIGHT 子句和编辑字符。



(2) 指针项(标识符8)必须是一个整型的初等数据项。

(3) 注意,在完成 STRING 语句时,接收项中未被送入的字符位置上保持原有内容,而不会自动设置空格。

(4) 所谓"强制语句",就是无条件的执行语句,IF 语句就不是强制语句,它是条件语句。

* §5.6 字符串分解语句(UNSTRING 语句)

(一) 将一个发送项的数据分别传送到多个接收项中,也就是将一个发送字符串拆成若干个接收字符串。它是 STRING 语句的逆操作。

假如数据项 A 的内容为:

DATE ␣␣ PRODUCT ␣ QUANTITY ␣

数据项 B,C,D 的长度分别为6,8,9个字节,执行:

UNSTRING A INTO B,C,D.

由左向右将字符从 A 送给 B,当 B 放满后,接着放到 C 中,C 满后再放 D 中。执行后 B 的内容为"DATE ␣␣",C 的内容为"PRODUCT ␣",D 的内容为"QUANTITY ␣"。

(二) 可以用 DELIMITED 短语来作分解时的定界符。自左而右累集字符,直到遇到指定的定界符为止。定界符左面的字符为传送内容,按 MOVE 语句的传送规则,接收项对送入的数据进行截断或补空格(或0)。如:

UNSTRING A DELIMITED BY 'T' INTO B, C.

B 的内容为"DA ␣␣␣␣",因为 T 以后的内容不传送,只送"DA",B 中的右端补空格。此时指针指在第三个字符位置(即"T"所在位置),这个位置已被检测过。接着从第四个字符开始向右扫描,直到遇第二个"T"为止,这段字符为"E ␣␣ PRODUC",应把它传送到第二个接收项 C 中,而 C 只有8个字节,按 MOVE 传送规则向左对齐,C 的内容为"E ␣␣ PRODU",字符 C 被截去。

如果向右扫描到发送项最右端仍未发现与定界符相同的字符,则传送以最右端的字符为截止界限。如:

UNSTRING A DELIMITED BY 'W' INTO B, C.

数据项A中找不到"W",则将整个字符串传送给B,而B只有6个字节,则B的内容为"DATE ␣␣",数据项C中没有被传送字符。

(三) 如果在DELIMITED BY后面加一个"ALL",表示定界符是长度不固定的一个常量。如:

```
UNSTRING A DELIMITED BY ALL '␣' INTO B, C.
```

表示定界符是一个或多个空格,把连续的一个或多个空格当作一个定界符。则B中内容为"DATE ␣␣"(右端已补二个空格),C的内容为"PRODUCT ␣"(已补一个空格)。

可以用多个定界符,用"OR"相连,表示只要满足其中之一即可。如:

```
UNSTRING  A  DELIMITED  BY ALL  SPACE  OR  '.␣' OR '␣' INTO  B, C.
```

表示遇到一个以上空格,或遇".␣"或"␣"均作为定界符,将其左边字符串向接收项传送,然后再从其后一个字符开始向右扫描检测,向第二个接收项传送。

(四) 有时接收项长度不够,不足以容纳传送给它的字符,我们可能希望知道已传送了多少字符。可以用COUNT短语(计数短语),将已发送的字符个数记入用户定义的记数数据项中。如:

```
UNSTRING  A  DELIMITED  BY  'T'  INTO  B  COUNT  IN W.
```

W是用户定义的数据名,它的值现在是"T"字符以左的字符个数,即2。如果数据项A中第一个字符就是与定界符相同的字符(如指定DELIMITED BY 'D')则W中值为0。注意记数数据项应为整数数值数据项(可以是COMP或DISPLAY用法的,见第七章),不能带编辑字符或P字符。COUNT短语必须和DELIMITED短语连用。

(五) 定界符存储短语(DELIMITER短语)

如果有多个定界符(如上面(三)所示),我们想知道送到各接收项中的字符是与那个定界符匹配的,可用DELIMITER短语。在传送一个字符串到接收项中时,同时将定界符送到某一数据项中存起来。

如:

```
UNSTRING  A  DELIMITED  BY  'T'  OR  ALL '␣' OR 'R'
     INTO  B  DELIMITER  IN  Q
           C  DELIMITER  IN  P
           D.
```

定界符有三个:T,␣和R。第一次遇到定界符"T",因此将T左边的"DA"传送给B,B的内容为"DA ␣␣␣␣"。同时将这个定界符"T"存到数据项Q中。第二次遇到两个空格(满足ALL"␣"条件),把它们前面的字符串("E")送给C,C的内容为"E ␣␣␣␣␣␣␣␣",同时将定界符两个空格送到P中。第三次遇到定界符"R",将"R"左面的字符串("P")送给D。D的内容为"P ␣␣␣␣␣␣␣␣␣"。由于D后面没有用DELIMITER短语,因此这个定界符("R")不存储。

当DELIMITER短语和COUNT短语一起使用时,DELIMITER短语应写在前面。

注意不要将DELIMITER IN(定界符存储)短语和DELIMITED BY(定界)短语混淆。

(六) 指针(POINTER)短语。如果希望从发送项某一指定的字符位置开始传送。可以用POINTER短语。如:

```
MOVE        5  TO    U
UNSTRING  A  INTO  C WITH  POINTER  U.
```

表示从A的第五个字符位置开始传送，C的内容为"␣␣PRODUC"。

一个UNSTRING语句只能有一个POINTER短语。

(七) 接收项记数(TALLYING)短语。用来记录实际接收传送的接收项项数。

```
MOVE  0  TO  N
UNSTRING  A  INTO  B,C,D  TALLYING  IN  N.
```

实际上B,C,D都从发送项A处收到字符，则N的值等于3。注意，是记录实际进行接收的接收项项数而不是字符个数。如果改为：

```
MOVE  0  TO  N.
MOVE  10  TO  U.
UNSTRING  A  INTO  B, C, D
                WITH  POINTER  U
                TALLYING  IN  N.
```

虽然语句中写了三个接收项，但由于指定从第十个字符位置开始传送，因此，只有B和C两个数据项收到字符。D没收到，故N的值应等于2而不是3。

TALLYING短语应写在POINTER短语后面。而这两个短语又应出现在DELIMITER和COUNT短语之后，因为TALLYING短语和POINTER短语是针对整个语句的而不是针对某一个接收项的。它们同样应是整数数值项不得带编辑字符和P字符。注意在使用TALLYING短语前应对TALLYING短语中的数据名赋初值(如上述MOVE 0 TO N)，每送完一个接收项，它的值就累加1。

(八) 溢出短语(OVERFLOW短语)，意思与STRING语句中所介绍的相同。什么情况算"溢出"呢?(1)当逻辑指针值小于1或大于发送项长度；(2) 当全部的接收项放不下发送项传送的内容时，就会产生"溢出"。如果有OVERFLOW短语，则在"溢出"时先执行UNSTRING语句再执行OVERFLOW短语中的强制语句，否则在执行UNSTRING语句后执行UNSTRING的下一语句。

(九) UNSTRING语句的一般格式为：

```
UNSTRING  标识符1
    [DELIMITED BY [ALL] {标识符2 / 常量1} [,OR [ALL] {标识符3 / 常量2}]…]
  INTO 标识符4,[,DELIMITER IN 标识符5][,COUNT IN 标识符6]
  [,标识符7 [DELIMITER IN 标识符8][,COUNT IN 标识符9]…]
    [WITH POINTER 标识符10] [TALLYING IN 标识符11]
      [;ON OVERFLOW  强制语句]
```

说明：

(1) 本语句中短语较多，根据需要选用，但应注意各短语的前后次序。

(2) 发送项可以是组合项，也可以是字符型初等项或字符编辑型数据项。

(3) 接收项可以是字母型的、字符型的、数值型的，但不能是编辑型数据项。如果是数值型项，则应该是 DISPLAY 用法的(见第七章)。PIC 字句描述中不得有 P 字符和编辑字符。当接收项是数值型数据项时，如果发送项传送的是数字字符，则把它作为无符号整数传送。

(4) 如接收项是非数值型数据项，则按非数值型初等项的 MOVE 语句规则传送。

(5) 如果接收项有 JUST RIGHT 子句，则发送和接收一律按右对齐处理。

(6) 如果发送项和接收项长度不等，按 MOVE 语句的规定对发送项截断或对接收项补空格或补零。

由于规定较多，使用本语句时一定要十分小心，防止出错。

* §5.7 检测语句(INSPECT 语句)

(一) INSPECT 语句可用来检查一个字符串中的字符：

(1) 累计一个指定的字符出现的次数。

(2) 用一个指定的字符去代替另一个指定的字符。

(3) 通过指定某些字符来限制上述检查的区间。

INSPECT 语句与上节 UNSTRING 语句有些类似，自左而右检查(扫描)被处理的数据项。也有定界短语、计数短语、指针短语等。例如

```
INSPECT SALES-RECORD TALLYING N FOR ALL '␣'.
```

本语句检查数据项 SALES-RECORD，找其中有无"␣"(空格)，找到一个就在计数器 N 中加1。

```
INSPECT SALES-RECORD REPLACING ALL SPACE BY ','.
```

找到每一个空格后，就以逗号替换该空格。

INSPECT 语句的基本格式为：

(1) INSPECT 标识符1 TALLYING 短语

(2) INSPECT 标识符1 REPLACING 短语

(3) INSPECT 标识符1 TALLYING 短语 REPLACING 短语

在此基础上还可以加写 BEFORE/AFTER 可选项。我们在下面逐个作简单介绍。

(二) BEFORE/AFTER 短语用来限制被检查的区域，如：

```
INSPECT SALES-RECORD TALLYING N FOR ALL '␣' BEFORE'.'.
```

用来检查在句点之前有几个空格，把空格计数存入 N 中，当然可以将 BEFORE"."改为 AFTER 'T'，检查在字符"T"之后有几个空格，等等。

BEFORE/AFTER 短语有些像上节的定界符短语，确定被检查的"界"。BEFORE/AFTER 短语中的定界符可以是一个字符。(如上述的"␣"或"T")，也可以是一个字符串，也可以是表意常量，或是一个标识符。

例如有：

```
INSPECT A RFPLACNIG ALL ZERO BY SPACE BEFORE'.'
```

假如 A 的内容为"000084.302"，执行 INSPECT 后将句点前所有的零换成空格，A 的内容变成"␣␣␣␣84.302"。

如果在被检查的字符串中找不到指定的定界符，则检查全部字符串。如将上面的 BE-

FORE '.'改为 BEFORE',',在 A 的内容中找不到逗号,则把整个字符串作为检测区域,将所有的零变成空格,此时 A 的内容变成"␣␣␣␣84.3␣2"。

(三) TALLYING 短语用来统计满足某种条件的字符的个数。如:

```
INSPECT  A  TALLYING  N  FOR  CHARACTERS BEFORE'.'.
```

统计在句点前有多少字符,把计数结果存放在数据项 N 中。

还可以用"LEADING"作条件的,如:

```
INSPECT  A  TALLYING  N  FOR  LEADING '0' BEFORE'.'.
```

意思是将 A 中的字符从最左的一个开始和"0"比较,如果有连续 n 个零,则 N 的值就是 n。如果 A 的内容为:"0012.34",第一个字符之后有两个连续的零,则 N 的值就是2。如果 A 的值是"0001200.34"出现两次连续的零,那么应统计哪一个?N 的值是3还是2?或是5?应统计最先出现的连续的零,LEADING 是"前边的"意思。因此 N 的值为3。如果 A 的内容为"123.4004",则 N 的值为0,因为零在句点之后,不在检测区之内。

一般对 N 赋以初值零,然后每找出一个零就向 N 加1。

由上可知,TALLYING 短语中有三种形式:

$$\left\{\begin{Bmatrix}\underline{\text{ALL}}\\\underline{\text{LEADING}}\end{Bmatrix}\begin{Bmatrix}\text{标识符}\\\text{常量}\end{Bmatrix}\right\}$$
$$\underline{\text{CHARACTERS}}$$

请读者考虑一下下面三个 INSPECT 语句中的 N 的值应是什么?

```
(1) INSPECT  A  TALLYING  N  FOR  ALL'*'AFTER'I'.
(2) INSPECT  A  TALLYING  N  FOR  LEADING'*' AFTER'I'.
(3) INSPECT  A  TALLYING  N  FOR  CHARACTERS  AFTER'I'.
```

设 A 的内容为"AT**F,***,T"。执行以上三个 INSPECT 语句后 N 的值如下:

(1) N=5 (2) N=2 (3) N=9

用 TALLYING 短语也可以统计指定的若干个字符出现的次数。如:

```
INSPECT  A  TALLYING  N  FOR  ALL '*', ALL ',', ALL 'I' AFTER 'I'.
```

将 I 之后的所有的"*",",",和"I"出现的次数都累计到 N 中,N=5+2+1=8。

也可以分别统计:

```
INSPECT  A  TALLYING
     N1  FOR  ALL'*' AFTER 'A'
     N2  FOR  ALL',' AFTER 'F'
     N3  FOR  ALL'I' BEFORE ','.
```

如果 A 的内容仍为"AT**F,***,T",则 N1的值为5,N2值为2,N3值为1。注意每个

$$\underline{\text{INSPECT}}\ \text{标识符1}\ \underline{\text{TALLYING}}$$
$$\left\{,\text{标识符2}\ \underline{\text{FOR}}\left\{,\left\{\begin{Bmatrix}\underline{\text{ALL}}\\\underline{\text{LEADING}}\end{Bmatrix}\begin{Bmatrix}\text{标识符3}\\\text{常量1}\end{Bmatrix}\right\}\left[\begin{Bmatrix}\underline{\text{BEFORE}}\\\underline{\text{AFTER}}\end{Bmatrix}\right.\right.\right.$$
$$\left.\left.\left.\underline{\text{INITIAL}}\begin{Bmatrix}\text{标识符4}\\\text{常量2}\end{Bmatrix}\right]\right\}\cdots\right\}\cdots$$

BEFORE/AFTER 短语只对其紧邻前边的字符统计有效。如果 N1后的 AFTER 'A' 和 N2 后面的 AFTER 'F' 不写，只写最后一个 BEFORE ","，则 N1的值是整个 A 的内容中所有的 '*' 的个数，N2的值是所有","的个数，而 N3的值是统计逗号之前的"T"字符的个数。

带 TALLYING 短语的 INSPECT 语句的一般格式见上页图所示。

(四) REPLACING 短语用指定的字符代替另一些指定的字符。如：

```
INSPECT  A  REPLACING ALL '0' BY SPACE
           ALL '.' BY ','
           ALL 'A' BY 'B'.
```

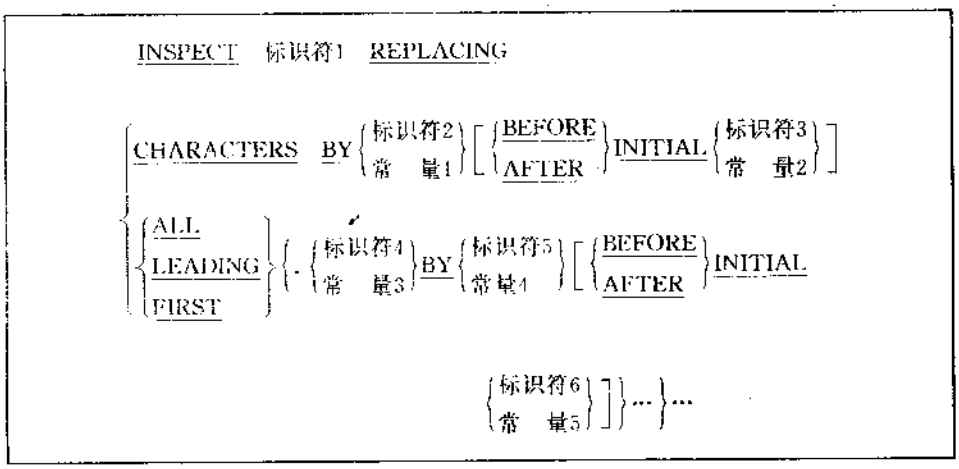
表示 A 中所有的"0"用空格代替，所有的句点"."用"逗号","代替，所有的"A"字符用"B"字符代替。如果 A 原来内容为"A000.12B"，执行 INSPECT 语句后 A 的内容变为："B␣␣␣,12B"。

REPLACING 短语中有一种"FIRST"的用法，如：

```
INSPECT A REPLACING FIRST '0' BY SPACE BEFORE '.'.
```

要求将句点以前的第一个零用空格代替，执行后 A 的内容为"A ␣00.12B"。这个 FIRST 选择在上面介绍的 TALLYING 短语中是不用的。

带 REPLACING 短语的 INSPECT 语句的一般格式为：



(五) COBOL 还允许在同一个 INSPECT 语句中同时用 TALLYING 短语和 REPLACING 短语，如：

```
INSPECT A TALLYING N FOR ALL 'L'
          REPLACING LEADING 'A' BY 'E' AFTER 'L.'
```

如果 A 原内容为"SALAMI"，则执行 INSPECT 语句后 A 内容变为"SALEMI"，N 的值为1。如果 A 原来内容为"ALABAMA"，执行后 A 内容变为"ALEBAMA"，N 的值为1。

同时带 TALLYING 短语和 REPLACING 短语的 INSPECT 语句的一般格式只是将前面两种格式综合在一起，读者可查本书附录中 ANSI COBOL 1974的语法格式，此处不再列出。

同时用 TALLYING 和 REPLACING 短语时，这两个短语的作用和分别写两个 INSPECT 语句时一样(其中第一个 INSPECT 语句只包括一个 TALLYING 短语，第二个只包括一个 REPLACING 短语)。上面的 INSPECT 语句等价于下面两个 INSPECT 语句：

INSPECT A TALLYING N FOR ALL 'L'.

INSPECT A REPLACING LEADING 'A' BY 'E' AFTER 'L'.

在写程序时最好将 TALLYING 和 REPLACING 短语分写在两个 INSPECT 语句中，以免搞错。其实在编译时，也是将它们拆成两个语句分别编译的。

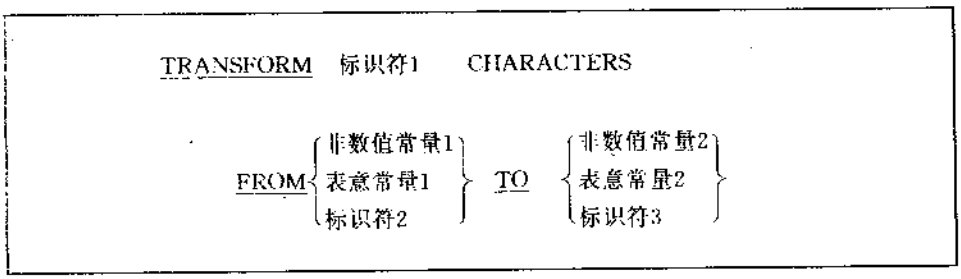
* §5.8 转换语句(TRANSFORM 语句)

如果想对某一数据项中的内容作某些字符转换，可用 TRANSFORM 语句来实现。如果数据项 T 的值原为"TOTAL-N"，今想将其中的"N"变为"M"，可写以下 TRANSFORM 语句：

TRANSFORM T FROM 'N' TO 'M'.

它的作用是将 T 中所有为"N"的字符变为"M"。

TRANSFORM 语句的一般格式为：



为叙述方便，将 TO 前面的数据项称为"被转换字符"，TO 后面的称为"转换字符"。如果"被转换字符"包含 n 个字符，而"转换字符"也有 n 个字符，则按顺序一一对应地进行转换。如果"转换字符"只有一个字符，则表示"被转换字符"所包含的各个字符都转换成"转换字符"中的字符。见下面几个例子。设 T 的原值为"TOTAL AMOUNT"。

语　句	T 转换后的值	说　明
TRANSFORM T FROM ' ' TO '-'	TOTAL-AMOUNT	所有空格⇒'-'
TRANSFORM T FROM 'AO' TO '12'	T2T1L-1M2UNT	'A'⇒'1','O'⇒'2'
TRANSFORM T FROM 'TNU' TO '*'	*O*AL AMO***	'T'⇒'*','N'⇒'*','U'⇒'*'
TRANSFORM T FROM A TO B (设 A 值已为'TO',B 的值已为'OF')	OFOAL AMFUNO	'T'⇒'O','O'⇒'F'
TRANSFORM T FROM 'A' TO SPACE	TO␣␣L␣␣MOUNT	'A'⇒' '

"转换字符"中所含字符个数不能大于"被转换字符"中含字符的个数，如 FROM 'ABC' TO 'TUXY' 是不正确的。如果"被转换字符"中含字符个数超过一个(二个或二个以上)，应要求"转换字符"中，含字符个数与之相同或只含一个字符。如写成：FROM 'XYZ' TO 'ABC' 或 FROM 'XYZ' TO 'A' 为正确，而 FROM 'XYZ' TO 'AB' 不正确。

“被转换字符”中不能含有两个以上相同的字符，如 FROM ‘XYX’ TO ‘ABC’，是不对的，因为有两个 X，分别对应 A 和 C，互相矛盾。而“转换字符”中可以包含二个以上相同的字符，如：FROM ‘ABCD’ TO ‘TTTA’，表示 A，B，C 都转换成 ‘T’，而 ‘D’ 转换成 ‘A’.

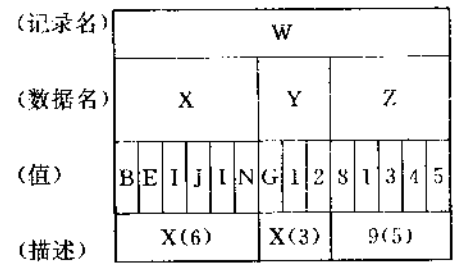
TRANSFORM 语句是非标准的语句，ANSI COBOL 1974中无此语句（它的功能可以由 INSPECT 语句来实现），但不少计算机系统（如 VAX 机等）有此语句，故作简单介绍。

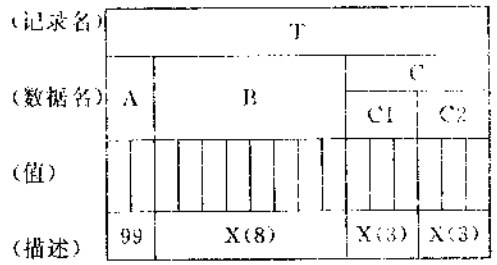
习　题

5.1　写出下面的 MOVE 语句执行后 B 中的值。如有不合法，指出不合法的原因。

	A 的值	B 的描述	B 的值
MOVE 1.5 TO B		99V99	
MOVE A TO B	69.82	99V99	
MOVE SPACE TO B		X(5)	
MOVE ‘ABC’ TO B		X(6)	
MOVE ‘NEW’ TO B		X(2)	
MOVE 1.2 TO B		99.99	
MOVE A TO B	182.65	9V9	
MOVE ‘ZERO’ TO B		X(5)	
MOVE ‘K-J’ TO B		$99.9	
MOVE 8.5 TO B		$$999	

5.2　写出下列传送的结果：





(1) MOVE　X　TO　B
(2) MOVE　Z　TO　A
(3) MOVE　X　TO　C
(4) MOVE　Z　TO　C
(5) MOVE　Z　TO　B
(6) MOVE　W　TO　T

5.3　如有以下两个组合项：

```
02  T1.
    03 X PIC X(20).
    03 Y.
      04 Y1 PIC 9(4).
      04 Y2 PIC 99V99.
    03 Z.
      04 Z1 PIC 99.
```

```
02  T2.
    04 Z.
        06 Z1 PIC 9V9.
        06 Z3 PIC 99.
    04 X.
        06 X1 PIC X(8).
        06 X2 PIC X(12).
```

04 Z2 PIC 9V9.　　04 W PIC X(15).

当执行 MOVE CORR T1 TO T2 时，哪些项是作为对应项传送的？写出它相当于那几个简单的 MOVE 语句。

5.4　什么叫数据名的受限？数据名与标识符有什么区别？什么叫两个组合项中的对应项？

5.5　写出下面计算结果存放在 C 中的情况。

ADD A,B GIVING C [ROUNDED]

计算结果	接收项 C 的描述	有无 ROUNDED	接收项 C 的内容
185.65	99V9	无	
83.687	99V99	有	
12.34	9V9	有	
63.463	99V99	有	

5.6　将本章5.2.2节中的例子（检查职工工资是否超过100元），写成完整的 COBOL 程序。

5.7　按下面要求写出语句，并写出 C 的 PIC 子句。

使 A+B⇒C。结果取二位小数，第三位四舍五入，如发生长度溢出，转到 G 段。

5.8　根据工资总额确定扣除率。如果已根据题意写出了以下过程部的 IF 语句，请用条件名来改写它，写出数据部和过程部的有关部分。GRSPAY 的值在0～2000元之间。

```
IF GRSPAY<1000.00
  IF GRSPAY>500.00
        MOVE 0.05 TO RETRMNT DEDUC
  ELSE MOVE 0.03 TO RETRMNT-DEDUC
ELSE MOVE 0.07 TO RETRMNT DEDUC.
```

工资总额 AMOUNT OF GROSS PAY (GRSPAY)	扣除率 RETIREMENT DEDUCTION RATE (RETRMNT-DEDUC)
等于或大于1000	0.07
大于500，小于1000	0.05
小于或等于500	0.03

5.9　分别按下面的要求，根据流程图（图5.8）写出程序。

（1）在 IF 语句中用简单条件。

（2）在 IF 语句中用复合条件。

（3）在 IF 语句中用条件名条件。

（X 在0～500范围之内）

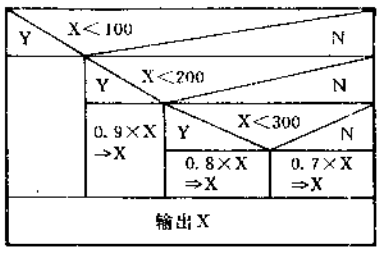


图 5.8

5.10　某银行有500个存户，要求统计出：

存款在100元以下的多少户?(不包括100元)

100元以上,500元以下的多少户?

500元以上,1000元以下的多少户?

1000元以上,5000元以下多少户?

5000元以上的多少户?

并分别统计出每一档中的平均存款额,和全部存户的平均存款额。

5.11 在§5.4中例5.7的程序中,如要求挑选的对象增加下列人员:

(1) 大学毕业,团员,女、二年工龄,志愿去东北者。

(2) 硕士学位,群众,男、三年工龄,志愿去西南者。

(3) 高中毕业,团员,男,四年工龄,志愿去南方者。请修改程序。

5.12 将§5.4中的例5.8画出流程图。

5.13 在§5.4的例5.9中,将打印的要求改为:每页打印40行零件数据细目行(不包括表头),打完40行应换一页,在每一张新页的头部都要打印出表头和列标题,然后再打印细目行。请修改补充该程序。

163

第六章　过程部之三

——执行语句(PERFORM 语句)

§6.1　执行语句的作用

在一个 COBOL 程序中，过程部中往往有一部分语句是需要多次执行的。因此，在程序的几个地方都需要重复出现这部分语句。如，计算存款的本利和，公式为 P1=P(1+R)。其中 P 为存款，R 为利率，P1 为年底的本利和。如果需要计算：

(1) P=1000，　　R=0.06　时 P1 的值。

(2) P=100，　　R=0.04　时 P1 的值。

则程序可写为：

```
A1. MOVE  0.06  TO  R.
    MOVE  1000  TO  P.
    ADD  1  TO R.
    MULTIPLY  P  BY  R  GIVING P1.
    MOVE  P1  TO  PRINTOUT.
    WRITE  PRINTOUT  AFTER  2.
A2. MOVE  0.04  TO  R.
    MOVE  100  TO  P.
    ADD  1  TO  R.
    MULTIPLY  P  BY  R  GIVING  P1.
    MOVE  P1  TO  PRINTOUT.
    WRITE  PRINTOUT  AFTER  2.
```

其中 A1 段求：P1=1000×(1+0.06)，A2 段求 P1=100×(1+0.04)。每段中虚线框内的部分是完全相同的，它是根据给定的 P 和 R 计算出 P1 并输出。如果需要多次求 P1，则需多次重复写出虚线框中的部分，显然这会使程序很冗长。

我们希望重复的部分只出现一次，即把重复的部分单独写成一段或一节(有一个段名或节名)，每次需要执行这部分语句时转去该段，执行完后转回来。这时就需要用到执行语句(PERFORM 语句)。

上面的例子可以改写成：

```
A1. MOVE  0.06  TO  R.
    MOVE  1000  TO  P.
    PERFORM C.
A2. MOVE 0.04 TO R.
    MOVE 100 TO P.
    PERFORM C.
    ⋮
C.  ADD  1  TO  R.
    MULTIPLY  P  BY  R  GIVING  P1.
    MOVE  P1  TO  PRINTOUT.
    WRITE  PRINTOUT  AFTER  2.
```

PERFORM 语句称为“执行语句”. “PERFORM C”的作用是：执行C段的语句。

以上程序段的执行过程是这样的：

（一）顺序执行A1段各语句。执行到PERFORM语句时，转到段名为C的段去执行。

（二）从C段的第一个语句开始执行，直到该段的最后一个语句为止。然后返回到调用它的PERFORM语句的下一个语句，在本例中为A2段。

（三）接着往下执行各语句(即A2段的语句)。当遇第二个PERFORM语句时，再次转去C段。执行完C段后再返回第二个PERFORM语句的下一语句。

其执行过程可以用图6.1表示。

①—②—③—④—⑤—⑥—⑦—⑧—⑨表示执行的顺序。

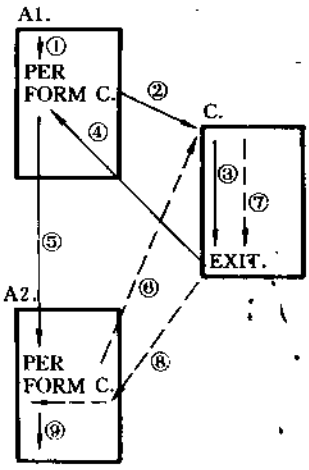


图 6.1

§6.2 执行语句的最基本的形式

PERFORM语句的最简单的形式为：

PERFORM　过程名

过程部中的节名、段名称为过程名。它大体相当于其它语言中的标号的作用，但以名字来表示。它标识过程中某一特定的部分，或者说它代表一段过程。上例中C段的名字C就是过程名。在简单的COBOL程序中，过程部下面不设节，直接由段组成。

注意，用PERFORM语句只能转到指定的节或段的开头，执行完该节(段)的全部语句后返回。不能直接转去执行某节(段)中某一个语句，也不能不执行完该节(段)就返回。简单地说，不能半截子进，半截子出。

有时，需要多次执行的部分不是一个段(或节)。而是若干个段(或节)，则应指明从哪一段(节)起到哪段(节)止。如下页首：

其执行过程为：

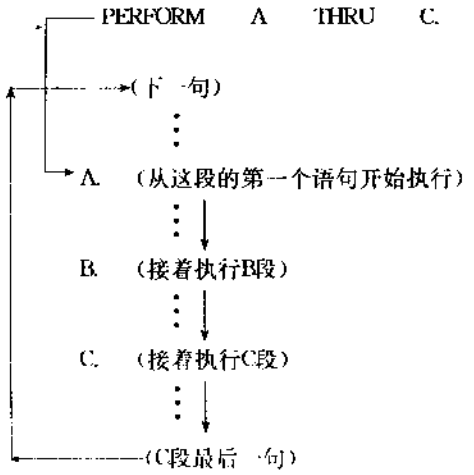
（一）遇PERFORM语句转去执行A段，执行到A段的最后一个句子。

（二）继续执行A段后面的各段(B段和C段)，直到C段的最后一个句子。

（三）返回到PERFORM的下一句，继续执行。

因此，PERFORM语句的一般格式可以扩充为：

PERFORM　过程名1[{THROUGH / THRU}过程名2]



如果不选方括弧中的可选项，则表示只执行一段(或节)。

如果过程名 1 和过程名 2 都是节名，则执行 PERFORM 语句时转向过程名 1 指出的节的第一个段中的第一个语句开始，并接着顺序执行该节的全部段，然后执行该节后面所有的节，直到过程名 2 指出的节的最后一段的最后一个句子执行完为止。最后返回 PERFORM 语句的下一个语句继续执行。

我们用图说明 PERFORM 语句的作用：

图 6.2 表示一个执行语句 PERFORM A THRU D 的执行过程。

程序执行各段的顺序是：

A⇒B⇒C⇒D⇒PERFORM 的下一句(T 段的第一句)，并接着往下执行。

图 6.3 表示有几个程序段，如果没有 PERFORM 语句，执行的顺序理应按①⇒②⇒③⇒①⇒⑤⇒⑥。而由于在 A 段中有 PERFORM B 语句，其顺序变为：①⇒②⇒①⇒③⇒④⇒⑤⇒⑥。

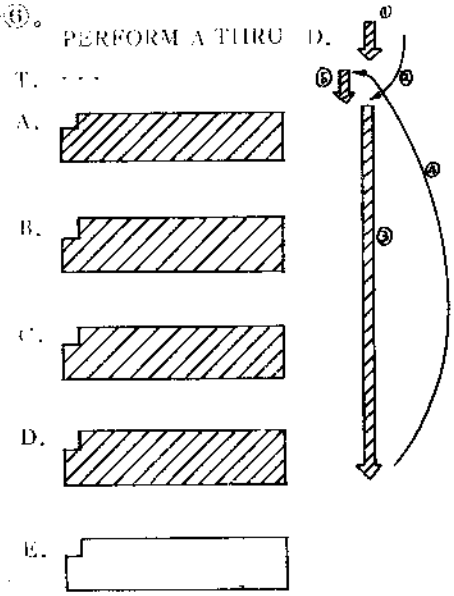


图 6.2

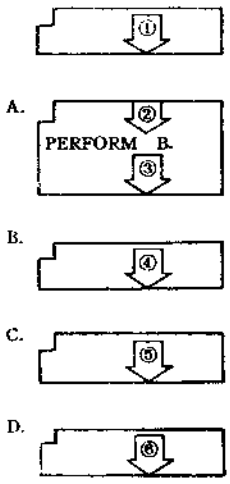


图 6.3

§6.3 执行语句的使用规则

(一) PERFORM 语句的嵌套。在一个执行语句所"调用"的语句序列中,又包括另一个执行语句。这叫"嵌套"。例如:

```
A.
    ⋮
    PERFORM C.
    ⋮
B.
    ⋮
C.
    ⋮
    PERFORM D.
    ⋮
D.
    ⋮
```

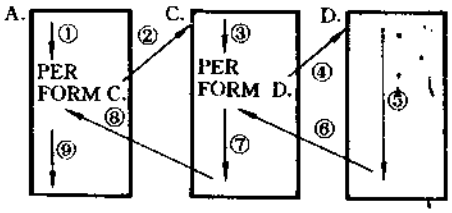


图 6.4

其执行过程见图 6.4。即:

① 开始执行 A 段语句。

② 遇"PERFORM C"语句转到 C 段。

③ 开始执行 C 段第一个语句。

④ 遇"PERFORM D",转到 D 段。

⑤ 执行 D 段中各语句。

⑥ 执行完 D 段最后一个句子后,转回"调用"它的 PERFORM 语句(即 PERFORM D)的下一句。

⑦ 继续执行 C 段中"PERFORM D"后面的语句。

⑧ 执行完 C 段所有语句后,转回"调用"它的 PERFOM 语句(PERFORM C)的下一句。

⑨ 继续执行 A 段中 PERFORM 下面的各语句。

⑩ 再执行 B 段、C 段、D 段,注意在执行 C 段时,包括一个"PERFORM D"语句。

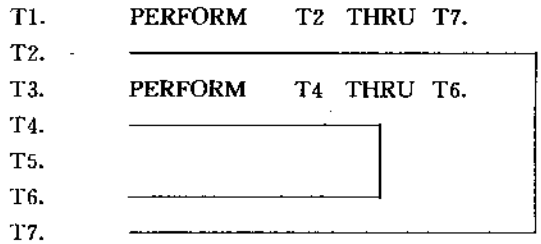
如果有以下一个程序片断:

```
A. DISPLAY  'A'.
   MOVE  'B'  TO  T.
   PERFORM  B.
   STOP  RUN.
B. DISPLAY  T.
   MOVE  'C'  TO  T.
   PERFORM  C.
C. DISPLAY  T.
```

这是一个 PERFORM 的嵌套。它相当于下列语句:

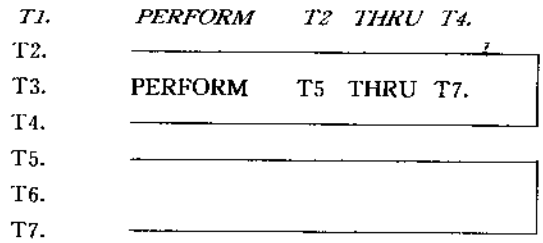
```
DISPLAY  'A'.      (显示 A)
MOVE  'B'  TO  T.
DISPLAY  T.        (显示 B)
MOVE  'C'  TO  T.
DISPLAY  T.        (显示 C)
STOP RUN.
```

注意，嵌套不能交叉。下面的嵌套是正确的：



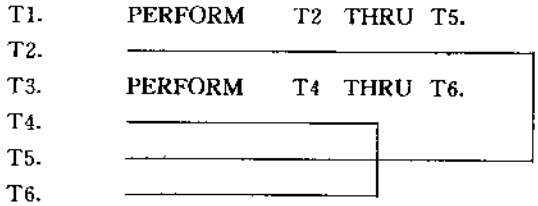
没有发生交叉。它的执行次序是：T1⇒T2⇒T3⇒T4⇒T5⇒T6⇒T4⇒T5⇒T6⇒T7⇒T2⇒T3…。T4～T6 全部在 T2～T7 之间。当后一个执行语句所“调用”的语句序列全部被包含在前一个执行语句所“调用”的语句序列范围之内，即一个被“调用”的语句序列全部套在另一个被“调用”的语句序列中，这样用法是合法的。

下面的用法也是正确的：



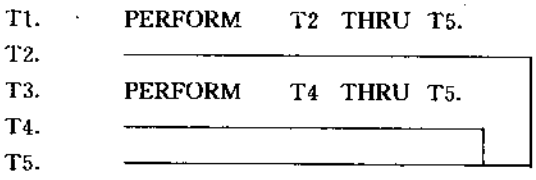
这时也没发生交叉。一个被“调用”的语句序列全部在另一个被“调用”的语句序列之外。它的执行次序是 T1⇒T2⇒T3⇒T5⇒T6⇒T7⇒T4⇒T2⇒T3⇒T5⇒T6⇒T7⇒T4⇒T5⇒T6⇒T7。

下面的用法不正确：



这时发生交叉。即后一个被“调用”的语句序列包含了前一个被“调用”的语句序列中一部分语句和该语句序列外的一部分语句。

以下的用法也不正确：



这时后一个被“调用”的语句序列还是包含了前一被“调用”的语句序列的最后一个段，即最

后一个段重叠。

综合起来，后一个被“调用”的语句序列或者全部套在前一个被“调用”的语句序列之中，或者全部在它之外是正确的。不允许二者交叉或有共同的终点（只有在最后一段中仅有一个EXIT 语句时才允许以这段作共同的终点）。但也有些计算机系统 COBOL 允许被 PERFORM 调用的语句序列交叉，用时可查所用的计算机系统 COBOL 手册。

请读者判断一下以下的用法是否正确？

```
T1.     PERFORM   T3  THRU  T5.
T2.     PERFORM   T4  THRU  T6.
T3.
T4.
T5.
T6.
```

这是允许的。因这两个 PERFORM 语句不存在嵌套关系，即第一个 PERFORM 语句执行完毕才遇到第二个 PERFORM 语句。其执行的次序是：

T1⇒T3⇒T4⇒T5⇒T2⇒T4⇒T5⇒T6⇒T3⇒T4⇒T5⇒T6

（第一个PERFORM作用）　（第二个PERFORM作用）

（二）在 PERFORM 语句所执行的语句序列中，可以含有转移语句，可以使流程转到语句序列之外，但一般应转回到此语句序列，以便最后能执行此语句序列的最后一个句子。执行完这个语句序列才能正常返回到“调用”它的 PERFORM 语句的下一语句，并使流程正确地继续下去。

下面的用法是正确的：

```
T1.     PERFORM   T3  THRU  T5.
T2.
T3.
T4.       GO    TO    T6
T5.
T6.
T7.       GO    TO    T5.
```

假设 T6 中不包含转移语句。则执行上述程序段的流程为：

T1⇒T3⇒T4⇒T6⇒T7⇒T5⇒T2⇒T3⇒T4⇒T6⇒T7⇒T5⇒T6⇒T7⇒T5……

（执行PERFORM的作用）

可以看出从 T3 直到 T5（中间虽然插入 T6，T7）是完整地实现了 RERFORM T3 THRU T5 的功能。最后返回到 PERFORM 的下一句，即 T2 段。

如果写成下面这样语句则 PERFORM T3 THRU T5 永远不会执行完，由于没有经过 T5，不能返回 PERFORM T3 THRU T5 的下一句。如果 T8 段中有一个 GO TO T5 语句使返回 T5，就可以了。

```
T1.     PERFORM   T3  THRU  T5.
T2.
```

```
T3.
T4.      GO    TO    T6
T5.
T6.
T7.      PERFORM    T9.
T8.
T9.
```

PERFORM 语句所执行的语句序列应包括一个完整的"过程"。下面的用法是错误的：

```
        PERFORM   X   THRU   Y.
        ⋮
        ⋮
X.      IF   A<B   GO   TO Y.
        COMPUTE   C=A-B
        GO   TO   Z.
Y.      COMPUTE   C=B-A
Z.      EXIT.
```

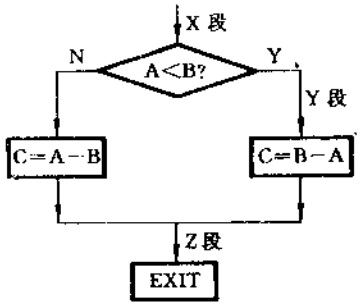


图 6.5

见图 6.5。因为 X 到 Y 并不是一个完整的过程。如果 A>B，则在执行 A－B⇒C 后，转到 Z 段，而没有经过 Y 段。这样就无法返回到 PERFORM 语句的下一语句。应写为：

```
PERFORM   X   THRU   Z.
```

这时，两个分支汇合在 Z 段，Z 是共同出口点。

§6.4 使用 PERFORM 语句实现循环

有时，需要利用一个 PERFORM 语句多次执行同一个语句序列（注意，不是在程序的不同地方分别利用几个 PERFORM 语句执行同一语句序列，而是一个 PERFORM 语句重复执行一个语句序列若干次）。这就意味着用 PERFORM 语句可以实现循环的功能。

如：

```
A1.      PERFORM    A5   THRU   A7     3    TIMES.
⋮
A5.
A6.                                    ←——重复 3 次
A7.
```

它表示执行 A5 到 A7 这部分语句序列共三次。其流程为：

A1⇒A5⇒A6⇒A7⇒A5⇒A6⇒A7⇒A5⇒A6⇒A7……

（第一次　第二次　第三次）

下面是两个程序片段，说明这种 PERFORM 语句的用法。

(1) A. PERFORM C 5 TIMES.
　　STOP RUN.

```
        ⋮
C.  READ ABC-FILE  AT  END  STOP  RUN.
    DISPLAY  ABC-EECORD.
```

执行 5 次 C 段。每执行一次 C 段，从 ABC-FILE 文件读入一个记录，并显示出这个记录(假设 ABC-RECORD 是 ABC-FILE 文件的记录，并已在数据部中说明)。共读入五个记录，显示出五个记录。

(2) 求本利和 P1=P(1+R)。已知本金 P，利息 R，求出五年中每年的本利和并显示出来。

```
AA.   MOVE  0.05  TO  R
      ADD  1  TO  R
      MOVE  100  TO  P
      PERFORM  CC  5  TIMES.
      STOP  RUN.
CC.   MULTIPLY  R  BY  P.
      DISPLAY  P.
```

开始，使 R=0.05，P=100，并求出 1+R⇒R。执行 CC 段五次。

第一次：P×R⇒P　结果为　$P_0(1+R)\Rightarrow P$

第二次：P×R⇒P　结果为　$P_0(1+R)^2\Rightarrow P$

第三次：P×R⇒P　结果为　$P_0(1+R)^3\Rightarrow P$

第四次：P×R⇒P　结果为　$P_0(1+R)^4\Rightarrow P$

第五次：P×R⇒P　结果为　$P_0(1+R)^5\Rightarrow P$

(P_0 为开始时的本金，即 100)

这种形式的 PERFORM 语句的一般格式为：

PERFORM 过程名 1 [{ THROUGH / THRU } 过程名 2] { 整数 / 标识符 } TIMES

说明：(1) 标识符应为整数数据项。如：

```
PERFORM  A  THRU  B  X  TIMES.
```

如果 X 的值为 10，则表示执行 A 到 B 段十次。如果 X 的值为零或负数，则此语句无效。

(2) 如果此标识符的值在执行语句序列中有变化，不会影响执行的次数。即以它开始时的值来 决定执行的次数。如

```
A.    MOVE  5  TO  N.
      MOVE  1  TO  M.
      PERFORM  A1  N  TIMES.
      DISPLAY  M.
      ⋮
A1.   MULTIPLY  N  BY  M
      SUBTRACT  1  FROM  N.
```

这是一个求 N! (1×2×3……×N)的程序片断。今使 N=5，求 5!。开始执行 PERFORM 语句时，N=5，即决定执行 A1 段五次。A1 段的作用是将 M * N⇒M，然后使 N 减 1，N−1=N。每一次执行 A1 段后 M 和 N 值如下：

	M	N
开始	1	5

第一次	5	4
第二次	5×4	3
第三次	5×4×3	2
第四次	5×4×3×2	1
第五次	5×4×3×2×1	0

尽管在执行 A1 段时 N 的值变化了，但并不影响执行的次数，即仍按开始执行 PERFORM 语句时 N 的值决定执行的次数(N=5)。此程序最后求得 M 的值为 120(5! 的值)。在执行 DISPLAY M 时显示出结果 120 来。

§6.5 执行语句的较复杂的形式

除了上一节介绍的指定循环次数的 PERFORM 语句以外，PERFORM 语句还有以下两种比较复杂的形式。

(一)

PERFORM 过程名 1 [{THROUGH / THRU} 过程名 2] UNTIL 条件

以上形式的 PERFORM 语句的作用是：反复执行指定的语句序列，直到给定的条件满足为止。可用流程图表示其流程(图 6.6)。

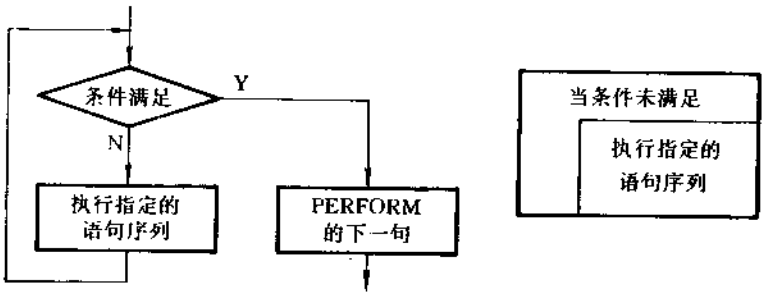


图 6.6

请注意：在执行 PERFORM 语句时，先判断指定的“条件”是否满足(为真)，若满足则执行 PERFORM 语句所指定的语句序列，如果一开始“条件”为假(不满足)，则一次也不执行指定的语句序列。因此，实际上它是一种“当型”循环结构。

例如，有以下一过程部的程序片断：

```
A.  MOVE  0  TO  T.
    MOVE  1  TO  N.
    PERFORM  B  UNTIL  N > 30.
    DISPLAY  T.
    STOP  RUN.
B.  ADD  N  TO  T.
    ADD  1  TO  N.
```

这个程序的作用是计算 1+2+3+…+30，即 $\sum_{n=1}^{30} n$。它的流程见图 6.7。

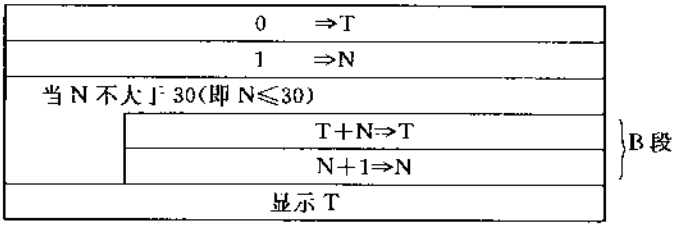


图 6.7

每次执行 B 段时 N 和 T 值变化如下：

第几次	T	N
未执行 B 前	0	1
第一次执行 B 段后	1	2
第二次执行 B 段后	1+2=3	3
⋮		⋮
第三十次执行 B 段后	1+2+3+…+30=465	31

在每一次执行 B 段之前，先判断给定的条件“N>30”是否成立？刚开始时 N=1，显然 N<30，也就是“N>30”为假，因此，执行一次 B 段(即 T+N⇒T，N+1→N)。此时 N=2，再判断一次“N>30”条件是否满足？显然还不满足，又执行一次 B 段(T+N→T，N+1⇒N)，此时，N=3，再判断“N>30”条件是否满足？……如此反复，在第三十次执行完 B 段后，N=31，此时显然“N>30”为真，不应再执行 B 段了，PERFORM 语句执行完毕。应接着执行 PERFORM 语句的下一语句。

可以看出，条件 N>30 中的 N 应在 B 段中加以改变。这里每执行一次 B 段，N 值加 1，因此执行 30 次 B 段后，N 的值就变为 31(N 的初值为 1)。显然，如果不使 N 的值改变，或者 N 变化的趋势不是趋向于满足给定的条件(例如使 N 每次减 1)，则永远不会满足 N>30 条件，循环会永无止境地继续下去，这叫“死循环”。

如下面的用法就会出现“死循环”。

```
A.    MOVE  0  TO  T
      MOVE  1  TO  N
      PERFORM  B  UNTIL  N > 30.
B.    ADD  N  TO  T.
```

N 始终等于 1 而不变化，永远不会满足 N>30。

(二)

PERFORM 过程名 1 [{ THROUGH / THRU } 过程名 2]

VARYING 标识符 1 FROM { 常 数 1 / 标识符 2 } BY { 常 数 2 / 标识符 3 } UNTIL 条件

例如：

```
PERFORM  T1  THRU  T2  VARYING  X  FROM  A
                       (循环变量)          (初值)
          BY  B  UNTIL  X > 5.
          (步长)        (条件)
```

它的作用是：执行 T1 到 T2 语句序列，X 是“循环变量”，是整型数据项。A 为初值，B 为步长，它们都是整数或整数数据项。其执行过程见图 6.8。

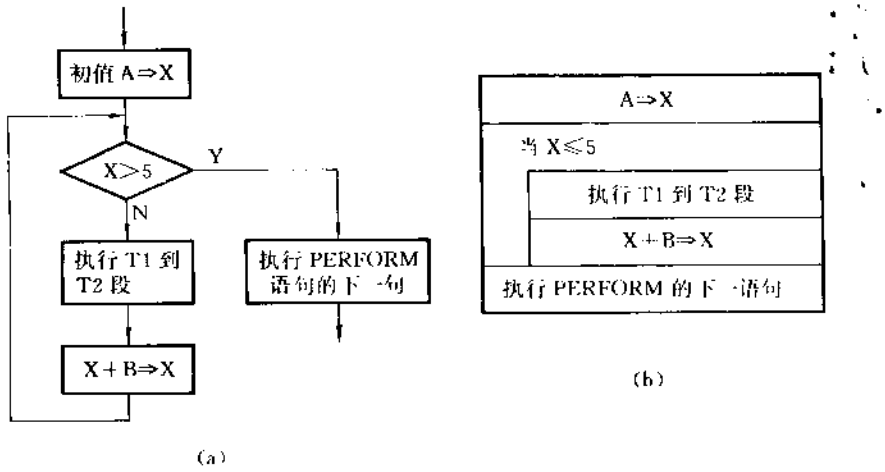


图 6.8

(1) 先把初值 A 送给 X，即循环变量 X 先取初值 A。

(2) 检查条件“X>5”是否满足？如果不满足，执行 T1 到 T2 段一次。

(3) 执行完一次 T1 到 T2 段后，X 按给定的步长 B 的值增值，X+B⇒X（若第一次开始时 X 等于 1，B=2，应使 1+2⇒X，X 的值由 1 变为 3）。

(4) 再判断 X>5？如不满足，再执行一次 T1 到 T2，然后 X 值再增加 2（第二次执行 T1 到 T2 段后 X 由 3 变成 5）。如此反复执行上述(2)，(3)，(4)，直到 X>5 为止。

(5) 当 X>5，PERFORM 语句执行完毕，不再执行 T1 到 T2，而执行 PERFORM 语句的下一个语句。

循环次数是由循环变量的值 A、变化增量（步长）B 和终止条件三者决定的。假设终止条件为 X>C，则循环次数 $r=\frac{C-A}{B}+1$。本例 A=1，B=2，C=5，计算出 $r=\frac{5-1}{2}+1=2+1=3$ 次。第一次执行完 T1～T2 段后，X 的值变为 3，第二次后，X 的值为 5，由于 X 不大于 5，还要再执行一次 T1～T2 段。当执行完第三次后，X 的值变为 7，X 已大于 5，不再执行 T1～T2 段了。共循环三次。

循环变量初值可为正值、负值或零，步长值也可为正值、负值，但不能为零。如果步长 B 值为零，则 X 值永远等于初值，而不会满足 X>5，出现“死循环”。

注意：循环变量 X 的值在每次循环中按步长增值（X+B⇒X），是自动完成的，不必在 T1～T2 段中再人为写出 ADD B TO X 语句。

UNTIL 后面的“条件”，可为任何条件，而不一定直接用到循环变量（上例中“条件”为 X>5，其中的 X 就是循环变量。“条件”中也可以不出现循环变量 X）。如：

```
A.  MOVE  0  TO  N.
    PERFORM  T1  THRU  T2  VARYING  X
      FROM  1  BY  5  UNTIL  N>10.
T1. ADD  1  TO  N.
    ⋮
T2.
```

此时，条件中并没有出现X，而出现了另一个数据名N，这是允许的。这时执行过程T1到T2多少次呢？共十一次。

	X	N
第一次后	6	1
第二次后	11	2
⋮	⋮	⋮
第十一次后	56	11

如果改写成：

```
    PERFORM  T1  THRU  T2  VARYING  X
      FROM  1  BY  2  UNTIL  N>10.
T1. COMPUTE  N = X * 2
    ⋮
T2.
```

此时执行过程T1到T2多少次呢？四次。执行完第四次T1到T2段后，N＝14，大于10，不再执行第五次。见下表所示。

第几次后	X		N＝2＊X
	增值前	增值后	（此处的X，是指增值前的X值，开始时X＝1）
1	1	3	2
2	3	5	6
3	5	7	10
4	7	9	14

一般说，“条件”是与控制变量有一定关系的。在上面两个PERFORM语句中，虽然在循环终止条件中未出现循环变量(X)，而出现了另一变量N，但在循环体T1～T2段中有一个使N值变化的语句，因此，通过N也能控制循环终止。如果在T1～T2段中，没有使N变化的语句，则无从通过N来控制循环结束，就会出现“无终止的循环”的情况。这是应注意的。因此对PERFORM语句的执行过程应精心设计。

§6.6 执行语句的多重循环形式

以上几种形式的执行语句是重复执行指定的语句序列若干次，也就是用PERFORM语句实现循环。上述PERFORM语句的循环是简单的循环，即一重循环(单层循环)，它根据一个给定的条件和有关循环参数决定循环的次数。在编程序中往往需要用到二重循环甚至三重循环。我们先看一具体例子：

```
PERFORM  T  VARYING  I  FROM  1  BY  1 UNTIL  I > 10
                   AFTER    J  FROM  2  BY  2 UNTIL  J > 8
```

这是一个二重循环的执行语句。AFTER 是“在……之后”的意思。即 J 先变，变到 J>8 时，I 才变一次。程序流程见图 6.9。其执行过程叙述如下：

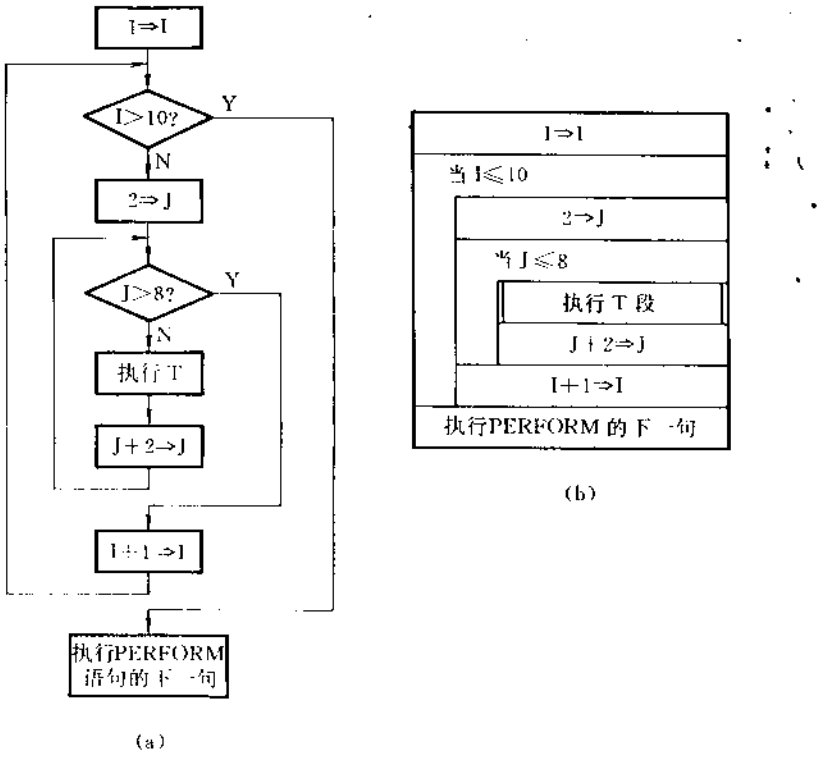


图 6.9

(1) 先使 I 取初值(等于 1)。

(2) 判断条件 I>10 满足了没有？如未满足此条件，则开始执行内循环。

(3) J 取它的初值(等于 2)。

(4) 判断 J>8？如未满足，则执行一次 T 段。

(5) J 按给定步长增值，J+2⇒J。

(6) 重复(4)、(5)，即再判 J>8？未满足，则再执行一次 T 段。再使 J+2⇒J。直到 J>8 为止。

(7) 满足 J>8(即 J 已按要求变化了一个周期)，使 I+1⇒I。这就是“AFTER…”的意思：在 J 变化一周期后 I 变化一次。

(8) 判断 I>10？如仍未满足，再使 J 重新取初值 2。然后重复上述(4)～(7)。直到 I 的值变化到 I>10 为止。

(9) I>10，则不再执行 T 段，而转去执行 PERFORM 语句的下一语句。即 PERFORM 语句已执行完毕。

由上述过程可知，I 取一个定值时，J 变化一个周期共要变化 4 次(J=2,4,6,8)。I 一共

要取 10 次值(I=1,2,3,4,5,6,7,8,9,10)因此共需执行 T 段 4×10=40 次。

J 为内循环的循环变量,I 为外循环的循环变量。内循环变量先变,外循环变量后变。学过 BASIC 的读者可以把它和下面的 BASIC 程序段类比:

```
FOR  I=1  TO  10  STEP  1
 FOR  J=2  TO  8  STEP  2
    (执行 T 段语句)
 NEXT  J
NEXT  I
```

【例 6.1】 打印九九乘法表。只写过程部中有关语句。

```
        PERFORM  A  VARYING  I  FROM  1  BY  1  UNTIL  I > 9
                   AFTER     J  FROM  1  BY  1  UNTIL  J > 9.
        ⋮
A.      COMPUTE  P  = I * J
        DISPLAY  I, '*', J, '=', P.
```

开始时　I=1，J=1，打印出　1*1=1

然后，　I=1，J=2，打印出　1*2=2

⋮　　⋮

I=1，J=9，打印出　1*9=9

此后,J+1=10,满足 J>9 条件,则使 I 由 1 变为 2,同时使 J 重取初值 1。接着执行 A 段 9 次,打印出:

```
2*1=2
2*2=4
⋮
2*9=18
```

然后,使 I+1⇒I,I 值等于 3,再继续下去。直到 I=9,J=9 为止。最后 J>9,I>9,不再执行 T 段,转去执行 PERFORM 的下一语句。

流程图见图 6.10。

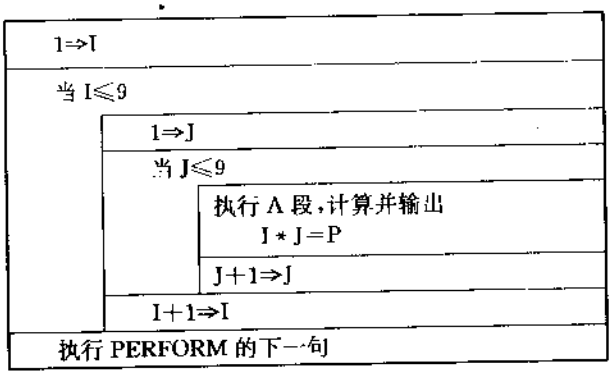


图　6.10

COBOL 允许用到三重循环,其一般格式可写为:

```
PERFORM 过程名 1 [THROUGH / THRU] 过程名 2
[VARYING 参数 1 FROM 初值 1 BY 步长 1 UNTIL 条件 1]
[AFTER   参数 2 FROM 初值 2 BY 步长 2 UNTIL 条件 2]
[AFTER   参数 3 FROM 初值 3 BY 步长 3 UNTIL 条件 3]
```

为了使读者容易理解，我们在上表中用了“初值”，“步长”，“参数”等名词，严格的格式在下册第九章中列出。上表中的参数 1、2、3 应是标识符，初值和步长可以是数值常量或标 识符。

以上格式的具体的 PERFORM 语句如下：

```
    PERFORM  A  VARYING  I  FROM  1  BY  1  UNTIL  I > 10
                AFTER    J  FROM  1  BY  1  UNTIL  J > 10
                AFTER    K  FROM  1  BY  1  UNTIL  K > 10.
     ⋮
A.  COMPUTE Z= ' I * J * K
    DISPLAY Z.
```

A 段共执行 10×10×10 次，即 1000 次。

§6.7 几种形式的执行语句的小结

我们共学过五种形式的 PERFORM 语句，为了使读者容易理解和接受，下面我们用具体的 PERFORM 语句来表示。

(一) PERFORM T1 THRU T2.

只能执行 T1 到 T2 部分 一次。好象把 T1 到 T2 这部分过程插入在 PERFORM 语句的位置上。这种形式比较简单。

(二) PERFORM T1 THRU T2 5 TIMES.

执行 T1 到 T2 共五次。循环的次数应是已知的。

(三) PERFORM T1 THRU T2 UNTIL N > 30.

执行 T1 到 T2 若干次。多少次？在写此语句时没有具体给定，只给出一个条件。在程序执行过程中根据 N 的值为判断“N>30”条件是否满足，来决定执行多少次。这种形式的执行语句比上面的灵活。

(四) PERFORM T1 THRU T2 VARYING X FROM 1 BY 2
UNTIL X > 30.

循环变量 X 能自动变化(每执行 一次 T1 到 T2 后，X 自动增值)，只要给的条件适当，X 的趋势应愈来愈趋近于满足条件(X>30)。不必在 T1 到 T2 段中再放入使 X 变化的语句。因此这种形式的 PERFORM 语句更灵活，功能更强，能解决 一般的循环问题。

(五) PERFORM T1 THRU T2
VARYING I FROM 1 BY 1 UNTIL I > 10
AFTER J FROM 2 BY 2 UNTIL J>8.

这是多重循环。在数据处理中常需用到几个层次的循环，用这种形式比较方便。

一般说，PERFORM 语句用来实现循环(第一种形式的 PERFORM 语句相当于循环次数为1次)。只是需要重复执行的部分在程序中的另外一个地方，按过程名找到这部分语句，循环执行若干次。见图6.11。

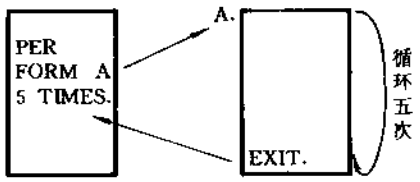


图 6.11

总之，PERFORM 的作用是：(1)“调用”本程序中某一语句序列。(2) 实现循环。

PERFORM 语句在 COBOL 程序设计中是很有用的。常常把程序的过程部写成“模块形式”。即，在“主模块”部分，只写若干个 PERFORM 语句，由它们分别“调用”有关的节或段。如下面形式：

```
PROCEDURE  DIVISION.
MAIN.
        PERFORM  A.
        PERFORM  B.
        PERFORM  C.
        ⋮
A.      …
B.      …
C.      …
```

用这种方法写程序，可使程序意思明了，作用明确，便于阅读和理解。见§6.9中例6.2。

§6.8 出口语句(EXIT 语句)

EXIT 语句提供了一组过程的公共出口，或者说它指出了被调用过程的逻辑终点。EXIT 语句不使计算机产生任何动作，它好比一个“桥梁”，能将被调用过程和调用过程的后续过程连接起来。从另一角度说，它的作用是提供一个段名，被 PERFORM 调用的语句序列由此公共汇集点(或称出口点)，返回到 PERFORM 的下一个语句上去。

例如：

```
(1)     PERFORM  A  THRU  B.        (2)     PERFORM  A.
        ⋮
    A. MOVE  1  TO  X.                  A. MOVE  1  TO  X.
       ADD  X  TO  Y.                      ADD  X  TO  Y.
    B. EXIT.
```

(1)和(2)的作用完全一样。对这个例子来说，EXIT 是可有可无的。但有时 EXIT 语句是不可缺少的(特别是被 PERFORM 调用的语句序列中有 IF 语句的情况下)。如：

```
        PERFORM  A  THRU  B.
        ⋮
A.  IF  X>Y  GO  TO  B.
    MOVE  X  TO  T.
    MOVE  Y  TO  X.
    MOVE  T  TO  Y.
B.  EXIT.
```

此时的 B 段中 EXIT 语句是不缺少的，它提供了一个公共出口，使 X>Y 时，执行 GO TO B，从而转到 B 段，也就是 EXIT 语句(出口)，接着执行 PERFORM 的下一语句。如果此时没有 EXIT 语句(无 B 段)，则 X>Y 时无处可转，程序不能正确执行。因此 B 段中的 EXIT 提供了 IF 语句的两个分支的共同出口点，不论那个分支，都从这里返回 PERFORM 的下一语句。见图6.12。

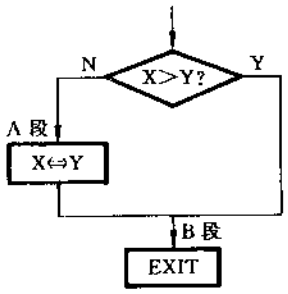


图 6.12

当然，上面的程序也可改写为：

```
PERFORM  A.
```

```
      ⋮
A.  IF  X  NOT  > Y
        MOVE  X  TO  T
        MOVE  Y  TO  X
        MOVE  T  TO  Y.
```

但 EXIT 语句用起来比较灵活、方便。一般用来作 PERFORM 语句调用的语句序列的出口。

注意，EXIT 必须是本段中唯一的语句，即该段中不允许有其它语句。EXIT 语句前必须有一段名。

§6.9 修改语句(ALTER 语句)

修改语句的一般格式：

```
ALTER  过程名1  TO  [PROCEED  TO]  过程名2
       [,过程名3  TO  [PROCEED  TO]过程名4]…
```

ALTER 语句用来改变 GO TO 的转向点。该语句使以过程名1，过程名3，…命名的各段中的 GO TO 语句的转向点分别被修改为过程名2，过程名4，……。注意，过程名1，过程名3，……各段只能由一条 GO TO 语句单独组成。

例如：

```
S.
  IF  X=5  PERFORM  PR-1
    IF  X=10  ALTER  PR-1  TO  B.
        ⋮
PR-1.
        GO  TO  A.
PR-2.   GO  TO  S.
              ⋮
A.
              ⋮
B.            ⋮
```

执行程序段，则首先判断 X=5是否满足，如果满足，则执行 PR-1段，无条件转向 A 段执行。若 X 不等于5，而等于10，第二个 IF 语句的条件满足，则将 PR-1段中的 GO TO 语句的转向点由 A 修改为 B。此后 PR-1段就成为：

```
PR-1.  GO  TO  B.
```

此后，在执行 S 段时，若 X=5，不再转去 A 段而转去 B 段。

§6.10 程序举例

【例 6.2】 由磁盘数据文件 A1.DAT 输入一批城市的名字、温度和湿度数据。要求打印出一张城市湿度清单。

清单格式要求每页开头打印出表头和栏目，然后打印各城市的数据，满25行后换一页。

每页都要打印表头和栏目。

程序如下：

```
IDENTIFICATION DIVISION.（标识部）
PROGRAM-ID.  EXAM62.
ENVIRONMENT  DIVISION.（环境部）
INPUT-OUTPUT  SECTION.
FILE-CONTROL.
      SELECT  INPUT-FILE ASSIGN  TO  AL.（输入文件）
      SELECT  OUTPUT-FILE ASSIGN  TO  P-FILE.（打印文件）
DATA    DIVISION.
FILE  SECTION.
FD  INPUT-FILE  LABEL  RECORD  IS  STANDARD.
01  IN-REC.（输入记录名）
    02 CITY  PIC X (10).（城市）
    02 TEMP  PIC S9(2).（温度）
    02 HUMID  PIC 9(3).（湿度）
FD OUTPUT-FILE LABEL RECORD IS STANDARD.
01 OUT-REC.（输出记录名）
    02 FILLER  PIC X.
    02 OUT-DATA  PIC X (80).
WORKING-STORAGE  SECTION.
77  IN END  PIC  X(3)  VALUE  SPACE.
77  LINE COUNT  PIC 9(2)  VALUE  25.（行数累计）
01  TITLE.（表头）
    02 FILLER PIC X(28) VALUE SPACE.
    02 FILLER PIC X(10) VALUE 'NATIONWIDE'.（全国范围）
    02 FILLER PIC X(10) VALUE SPACE.
    02 FILLER PIC X(11) VALUE 'TEMPERATURE'.（温度）
01  SUBTITLE.（小标题）
    02 PIC X(20) VALUE SPACE.
    02 PIC X(9) VALUE 'CITY NAME'.（城市名）
    02 PIC X12) VALUE SPACE.
    02 PIC X(11) VALUE 'TEMPERATURE'.（温度）
    02 PIC X(8) VALUE SPACE.
    02 PIC X(8) VALUE 'HUMIDITY'.（湿度）
01  DETAILS.（细目）
    02 FILLER PIC X(20) VALUE SPACE.
    02 CITY PIC X(10).
    02 FILLER PIC X(15) VALUE SPACE.
    02 TEMP PIC --- -.
    02 FILLER PIC X(2) VALUE 'C'.
    02 FILLER PIC X(11) VALUE SPACE.
    02 HUMID PIC ZZ9.
    02 FILLER PIC X(2) VALUE '%'.
PROCEDURE  DIVISION.（过程部）
START-RUN.（"开始运行"段）
     OPEN       INPUT  INPUT-FILE
                OUTPUT  OUTPUT-FILE.
     MOVE       SPACE  TO  IN-END.（对 IN-END 项冲空）
READ-RECORD.（"读记录"段）
     READ       INPUT-FILE
                AT END MOVE 'END' TO IN-END.
     PERFORM IN-PROCESSING UNTIL IN-END = 'END'.
```

```
    CLOSE INPUT-FILE, OUTPUT-FILE.
    STOP RUN.
IN-PROCESSING. ("处理输入记录"段)
    IF    LINE-COUNT  -25
          MOVE SPACE TO OUT-DATA      (对 OUT-DATA 冲空)
          WRITE OUT-REC BEFORE PAGE     (先输出一个空行,再换页)
          MOVE SPACE TO OUT-DATA
          WRITE OUT REC BEFORE 1      (输出一空行)
          MOVE TITLE TO OUT-DATA
          WRITE OUT-REC BEFORE 2      (输出表头,后空一行)
          MOVE SUBTITLE TO OUT DATA
          WRITE OUT-REC BEFORE 2      (输出小标题,后空一行)
          MOVE ZERO TO LINE-COUNT     (使行计数冲零,从0行起重新计数)
    ELSE
          MOVE CORR IN-REC TO DETAILS     (对应传送)
          MOVE DETAILS TO OUT-DATA
          WRITE OUT-REC BEFORE 1
          ADD 1 TO LINE-COUNT
          READ INPUT FILE AT END MOVE 'END' TO IN END.
```

为简单起见,我们假设磁盘数据文件 A1.DAT 中有两个城市的数据,数据形式如下:

```
BEIJING     13035
SHANGHAI 24042
```

输出文件 P-FILE.DAT 文件中的数据形式如下:

NATIONWIDE　　TEMPERATURE　　(全国范围温度)

CITY NAME(城市名)	TEMPERATURE(温度)	HUMIDITY(湿度)
BEIJING	13C	35%
SHANGHAI	24C	42%

可以将此文件中内容打印出来。

程序执行情况见图6.13。打开文件后,读入一个记录,如果遇到"文件结束标志",则将"END"传送给 IN-END(IN-END 已经在数据部中定为3个字符长度)。如果遇到的不是"文件结束标志",那就是有效的记录。PERFORM 语句指定转去执行 IN-PROCESSING 段。

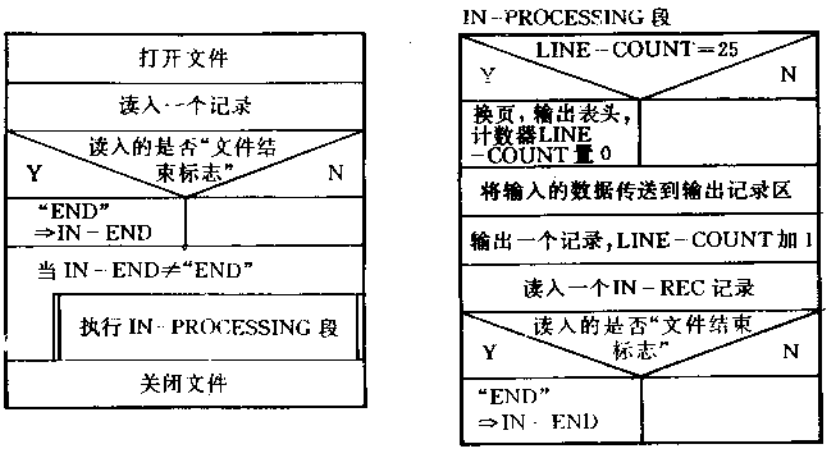


图　6.13

在 IN-PROCESSING 段中，首先检查数据项 LINE-COUNT 的值是否等于25?，今在数据部中已使它的初值为25，因此将 OUT-DATA 冲空，输出 OUT-REC 记录区的内容，因前面刚把 OUT-DATA 冲空，这样在输出 OUT-REC 时，我们就得到一个空行，然后换页。换页后再输出一个空行，接着输出表头和小标题，并使 LINE-COUNT 值变为零(因一页开始，故行计数应从零开始重新计数)。把输入的有关数据用对应传送的方法送到 DETAILS 中(对它进行编辑并加上必要的文字说明)，再送到 OUT-DATA，然后输出，即输出一个城市的温度和湿度记录。然后使计数项 LINE-COUNT 加1 (因已输出一行)。再读下一个记录。至此，已执行完一次 IN-PROCESSING 段。由于 PERFORM 语句给定的条件 IN-END="END"未满足，因此还要执行一次 IN-PROCESSING 段。此时 LINE-COUNT 的值为1，而不等于25，因此，不换页和不输出表头，直接执行对应传送，将输入的数据经编辑后再输出一行(另一城市的记录)。LINE-COUNT 变为2。如此反复，直到输出完25行以后，LINE-COUNT 值变为25。再读入下一个记录，先输出一空行，然后换页打印表头，再接着输出温度湿度记录。直到记录读完，遇到"文件结束标志"，则执行 AT END 子句，将"END"三个字符传送给 IN-END。此时，已满足 PERFORM 语句中 UNTIL 后面的条件(IN-END="END")，因此不再执行 IN-PROCESSING 段，而关闭文件和停止运行。

图6.13所示的流程图中，左面的是主流程图，它表示程序的总的流程状况，其中下部的当型循环结构表示的是 PERFORM 语句的流程。"执行 IN-PROCESSING"框的上端画有双线，这表示它不是一个简单的操作语句，而是一个模块。这个模块的流程另行画出在图6.13右面。如果一个程序中包含多个模块，都要一一分别画出。这样的方法可以清楚地看到程序的主流程，又能详细地看到每一个模块内部的操作步骤。如果不按模块分别画，而把全部流程画在一起，会使流程图庞杂，层次不分明，主次不分，难以弄清楚流程。

【例 6.3】 读入一组产品销售记录，每读入一个记录，计算出销售总额(数量×单价)。然后打印出该产品的全部数据。数据形式如下：

```
9 2 1 2 1 2 0 0 0 0 0 1 6 6 6 6 0 1 0 0 0 1 0 0
9 2 1 3 1 3 0 0 0 0 0 2 8 8 8 8 0 2 0 0 0 1 0 0
└──┘ └──┘ └──┘ └──┘ └──┘
 日期   产品号  顾客号  数量   单价
```

程序如下：

```
IDENTIFICATION DIVISION. (标识部)
PROGRAM-ID. EXAM 6.3.
ENVIRONMENT  DIVISION. (环境部)
INPUT OUTPUT SECTION.
FILE-CONTROL.
    SELECT  INPUT-FILE  ASSIGN  TO  FILE1.
    SELECT  OUTPUT-FILE  ASSIGN  TO  P-FILE. (打印文件)
DATA DIVISION. (数据部)
FILE SECTION.
FD  INPUT-FILE LABEL RECORD IS STANDARD.
01  IN-REC.
    02 DATEE  PIC 9(6). (日期)
    02 PRODUCT-CODE  PIC 9(6). (产品号)
    02 CUSTOMER-CODE PIC 9(4). (顾客号)
    02 QUANTITY      PIC 9(4). (数量)
    02 UNIT-PRICE    PIC 9(4). (单价)
```

```
FD OUTPUT-FILE  LABEL  RECORD  IS  STANDARD.
01  OUT REC.
    02  FILLER  PIC  X.
    02  DATEE  PIC 99B99B99.(日期)
    02  FILLER  PIC X(5).
    02  PRODUCT-CODE  PIC 9(6).(产品号)
    02  FILLER  PIC  X(5).
    02  CUSTOMER CODE  PIC  9(4).(顾客号)
    02  FILLER  PIC  X(5).
    02  QUANTITY  PIC  ZZZ9.(数量)
    02  FILLER  PIC  X(5).
    02  UNIT-PRICE  PIC  $(5).(单价)
    02  FILLER  PIC  X(5).
    02  SALES-VALUE  PIC $(8).(销售金额)
PROCEDURE    DIVISION.(过程部)
MAIN.(主模块)
      PERFORM     PREPARATION.(预备)
      PERFORM     RECORD-PROCESSING(记录处理)
                  UNTIL CUSTOMER-CODE OF IN-REC = 9999.
      PERFORM WINDING-UP.(关闭)
      STOP RUN.
PREPARATION.("预备"段)
      OPEN  INPUT     INPUT-FILE
            OUTPUT  OUTPUT-FILE.
      READ  INPUT-FILE(读入一个记录)
            AT END DISPLAY 'NOTHING READ'
                   MOVE  9999  TO CUSTOMER-CODE  OF IN REC.
RECORD-PROCESSING.("记录处理"段)
      MOVE SPACES TO OUT-REC.
      MOVE CORR IN-REC TO OUT-REC.
      COMPUTE SALES-VALUE--
              QUANTITY OF IN-REC * UNIT-PRICE OF IN REC.(计算销售额)
      WRITE OUT-REC AFTER 1.(输出记录)
      READ INPUT FILE(读入一个记录)
           AT END MOVE 9999 TO CUSTOMER-CODE OF IN REC.
WINDING-UP.
      CLOSE INPUT FILE, OUTPUT-FILE.(关闭文件)
```

输出数据形式如下：

```
92 12 12     000001     6666     100     $100     $10000
92 13 13     000002     8888     200     $100     $20000
```

本程序的特点是：采用模块式程序设计，将程序应执行的各个部分功能分别写成若干个段。然后在主模块部分(MAIN 段)，只简单地写出几个 PERFORM 语句即可。主模块如同一个"调度"，先后"调用"几段执行，然后结束。我们用有含义的英文字作段名，从 MAIN 段中可以明白地看到每一个 PERFORM 语句的作用：预备→记录处理→结束。主模块的流程图见图6.14(a)。

执行 PREPARATION 段，读入一个记录，如遇文件结束记录，将9999送到 CUSTOMER-CODE 中（见图6.14(b)）。然后在主模块中判断 CUSTOMER-CODE 是否等于9999（假设顾客号的值不会是9999，而最多只达百位数如0123），如不等于9999，则执行一次 RECORD-PROCESSING 段，用对应项传送的方法，将 IN-REC 中与 OUT-REC 中数据项

同名的数据项(DATEE,PRODUCT-CODE,CUSTOMER-CODE,QUANTITY, UNIT-PRICE)从IN-REC传送到OUT-REC的相应项中。然后计算SALES-VALUE(销售额),并输出一个记录。再读入一个记录,如遇文件"结束标志",将9999送到CUSTOMER-CODE。见图6.14(c)。

从图6.14(a)中可以看出,如果刚读入的不是"文件结束标志",则COUSTOMER-CODE的值不会是9999,应再执行一次RECORD-PROCESSING,如此反复。每读入一个记录,输出一个相应的记录。直到全部数据记录读完。最后遇到"文件结束标志",根据读语句中AT END子句的规定,CUSTOMER-CODE的值变为9999。此时,程序里第二个PERFORM语句中UNTIL后面的条件满足,不再执行RECORD-PROCESSING段。转而执行WINDING-UP段,见图6.14(d),程序结束。

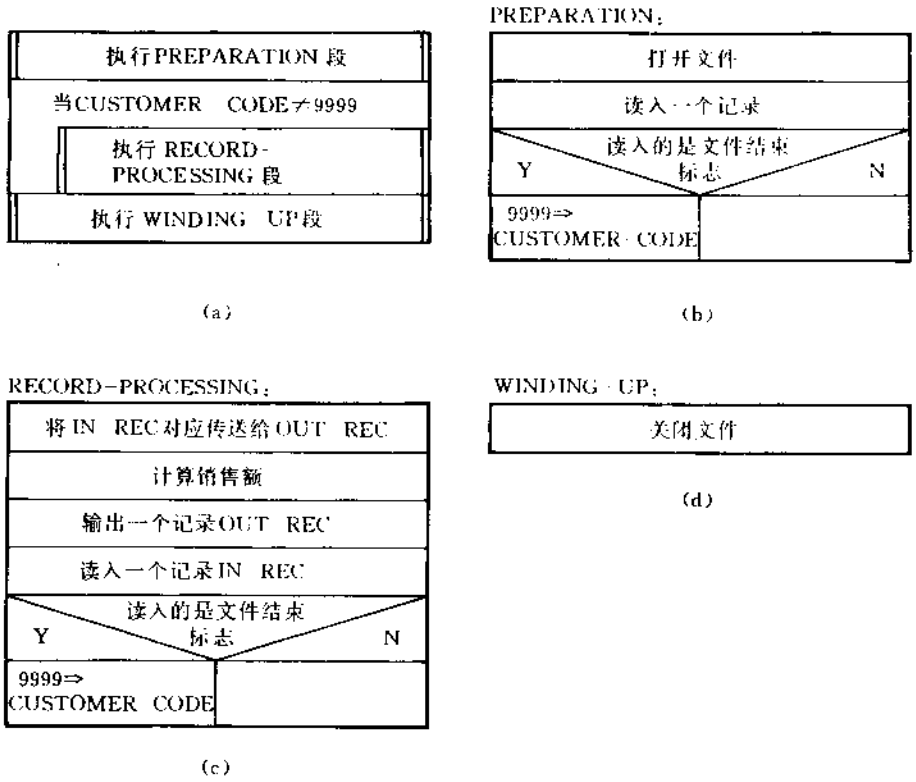


图 6.14

由于这是一个练习程序,所以打印格式比较简单,没有打印"表头"和小标题部分。读者可以根据已学过的知识,修改程序,使它能打印出"表头"和小标题。

请读者写出输出的格式。

【例 6.4】 赊购外贸货物,假如已知货款和合同规定的付款月数和月利率,每月应付的款项为:

$$\text{应付款}=\frac{\text{赊购的款项}}{\text{付款的月数}}+\text{欠款}\times\text{利息}$$

如果向某公司赊购的货款为18000元，月利率为0.012，规定一年半付清。则

第一个月末的应付款$=\frac{18000}{18}+0.012\times18000=1216$元

第二个月末的应付款$=\frac{18000}{18}+0.012\times(18000-1000)=1204$元

（尚欠款部分）

每月付的款＝每月应付的欠款部分（赊购货款÷分期付款的月数）＋月利息

其中前者为固定的值，后者为可变的值，它随尚欠款的部分减少而减少。

今有若干个单位赊购货物，希望为每一个单位打印出一张每月付款清单，其格式为：

NAME：WANG　　SCHEDULE OF PAYMENTS（月付款表）

ORIGINAL AMOUNT $18,000.00

MONTH (月份)	INTEREST (利息)	TOTAL PAYMENT (总付款)	UNPAID BALANCE (尚欠款)
1	216.00	1216.00	17000.00
2	204.00	1204.00	16000.00
⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮
12			

程序如下：

```
IDENTIFICATION  DIVISION.  (标识部)
PROGRAM ID. EXAM64.
ENVIRONMENT  DIVISION.(环境部)
INPUT-OUTPUT  SECTION.
FILE-CONTROL.
    SELECT  LP-FILE  ASSIGN  TO  P-FILE.(打印文件)
    SELECT  INPUT-FILE  ASSIGN  TO  IN-FILE.(输入文件)
DATA  DIVISION.(数据部)
FILE  SECTION.
FD INPUT-FILE  LABEL RECORD IS STANDARD.
01  IN-RECORD.
    02  NAME  PIC X(20).  (名字)
    02  SALEVALUE  PIC 9(5)V99.  (赊购额)
    02  NUMB OF MONTHS  PIC 99.  (规定付款的月数)
FD  LP-FILE  LABEL  RECORD  IS STANDARD.
01  LINEOUT  PIC  X(137).
WORKING-STORAGE SECTION.
77  AMOUNT  PIC S9(5)V9(4).  (金额)
77  MONTH  PIC  S99.          (月)
77  PAYMENT  PIC  S9(4)V99.  (应付款)
77  INTEREST  PIC  S9(3)V9(4).  (利息)
77  CONSTAN  PIC  S9(4)V9(4).  (每月应付的固定部分)
77  END-DATA  PIC X(3)  VALUE  SPACE.
01  EDITOR.  (打印的各月付款的信息)
    02  FILLER  PIC  X.
    02  ED-MONTH  PIC  ZZZ9.
    02  FILLER  PIC  X(7)  VALUE  SPACES.
    02  ED-INTEREST  PIC  Z(4).99.
    02  FILLER  PIC  X(10)  VALUE  SPACES.
```

```
    02  ED-PAYMENT  PIC  Z(5).99
    02  FILLER  PIC  X(8)  VALUE  SPACES.
    02  BALANCE  PIC  -(6).99.    (尚欠款)
01  HEADER-1.  (要打印的第一行的文字)
    02  FILLER  PIC  X(6)  VALUE  'NAME:'.
    02  HEAD-NAME  PIC  X(14)  VALUE  SPACES
    02  TITLE-1  PIC  X(20)  VALUE  'SCHEDULE  OF  PAYME  NTS'.  (注:本行紧
        接上行)
01  HEADER-2.  (第二行的文字)
    02  FILLER  PIC  X(18)  VALUE  SPACES.
    02  TITLE-2  PIC  X(15)  VALUE  'ORIGINAL AMOUNT'.
    02  FILLER  PIC  X(3)  VALUE  SPACES.
    02  HEAD-AMOUNT  PIC  $$$.$$$.99.
01  HEADER-3.  (第三行的文字)
    02  FILLER  PIC  X(6)  VALUE  'MONTH'.
    02  FILLER  PIC  X(6)  VALUE  SPACES.
    02  FILLER  PIC  X(8)  VALUE  'INTEREST'.
    02  FILLER  PIC  X(6)  VALUE  SPACES.
    02  FILLER  PIC  X(13)  VALUE  'TOTAL  PAYMENT'.
    02  FILLER  PIC  X(6)  VALUE  SPACES.
    02  FILLER  PIC  X(14)  VALUE  'UNPAID  BALANCE'.
PROCEDURE  DIVISION.    (过程部)
MAIN.
    PERFORM  A.
    PERFORM  B  THRU  C  UNTIL  END-DATA = 'END'.
    STOP  RUN.
A.  OPEN  INPUT  INPUT-FILE
              OUTPUT  LP-FILE.
    READ  INPUT-FILE
          AT  END  MOVE  'END'  TO  END-DATA
                   CLOSE INPUT-FILE, LP-FILE.
B.  MOVE  SPACE  TO  LINEOUT.
    WRITE  LINEOUT  AFTER  PAGE.
    MOVE NAME TO HEAD-NAME.
    MOVE SALEVALUE(赊购额) TO AMOUNT, HEAD-AMOUNT.
    WRITE LINEOUT FROM HEADER-1 AFTER 1.
                          (打印表格第一行)
    WRITE LINEOUT FROM HEADER-2  AFTER 1.
                          (打印表格第二行)
    WRITE LINEOUT FROM HEADER-3  AFTER 1.
                          (打印表格第三行)
C.
    DIVIDE NUMB-OF-MONTHS  INTO  AMOUNT
           GIVING CONSTAN.  (计算月固定付款数)
    MOVE  1  TO  MONTH.
    PERFORM  D  THRU  E
             UNTIL  MONTH > NUMB-OF-MONTHS.
    READ  INPUT-FILE
          AT  END  MOVE  'END' TO  END-DATA
    CLOSE  INPUT FILE, LP-FILE.
D.  MULTIPLY  0.012  BY  AMOUNT  GIVING  INTEREST.
                          (计算本月应付利息)
    ADD  CONSTAN  INTEREST
       GIVING  PAYMENT  ROUNDED.  (计算本月应付款)
```

```
    SUBTRACT CONSTAN  FROM  AMOUNT.  (计算尚欠款)
E.  MOVE  MONTH  TO  ED-MONTH.
    MOVE  INTEREST  TO  ED-INTEREST.
    MOVE  PAYMENT  TO  ED-PAYMENT.
    MOVE  AMOUNT  TO  BALANCE.
    WRITE  LINEOUT  FROM  EDITOR  AFTER  1.
    ADD  1  TO  MONTH.
```

说明：(1) 数据文件 IN-FILE 中记录形式为每个赊购人的姓名、赊购额和应付款月数。

WANG	180000	018
20	7	2
(名)	(赊销额)	(月数)

(2) BALANCE(尚欠款)用 PIC—(6).99描述，而 BALANCE 理应为正或零，那么，为什么要用 PIC—(6)呢?这是为了万一程序发生某些错误而计算出尚欠款值为负时，可以打印出负号，这是编程序时考虑的一个检查错误的方法。如不用负号，出现负值时，打印结果仍为无符号的数，不易发现错误。

(3) 本例在过程部中也是使用 PERFORM 语句，形成模块方式，段名用了 A，B，C……，而没有用有含义的英文字，只为了简单，因为本程序仅为练习用。至于在实际应用时是用英文字、或汉语拼音、或用简单的代码符号(如 A，B，C…)，完全由读者自便。

(4) A 段的作用是打开文件，并读入第一个记录。B 段作用是使每一个单位的付款清单都从新的一页开始打印，并打印表头。C 段作用是计算每月固定的应付款数，并计算每个月的应付利息、应付款总数和尚欠款。C 段中又包含了一个 PERFORM 语句，执行 D 到 E 段的次数为 NUMB-OF-MONTHS 次。譬如给定的 NUMB-OF-MONTHS 的值为18，则共执行 D 到 E 段18次，这是由 UNTIL MONTH>NUMB-OF-MONTHS 短语来控制的。在 C 段中先使 MONTH 值为1，以后每执行一次 E 段就使 MONTH 加1，直到满足 MONTH>NUMB-OF-MONTH 就不再执行 D 到 E 段程序。

(5) E 段也可以改用 MOVE CORR 形式的对应项传送方式，可以少写几个 MOVE 语句，但数据部中的数据名应作相应修改。

假如在一次运行中，所给的输入数据如下：

```
WANG        180000018
TAN         200000012
CHANG       120000024
LI          451200036
LING        015000010
FUNG        055000006
```

程序运行时输出结果如下：

NAME:WANG

SCHEDULE OF PAYMENTS

ORIGINAL AMOUNT $18,000.00

MONTH	INTEREST	TOTAL PAYMENT	UNPAID BALANCE
1	216.00	1216.00	17000.00
2	204.00	1204.00	16000.00
3	192.00	1192.00	15000.00
4	180.00	1180.00	14000.00

5	168.00	1168.00	13000.00
6	156.00	1156.00	12000.00
7	144.00	1144.00	11000.00
8	132.00	1132.00	10000.00
9	120.00	1120.00	9000.00
10	108.00	1108.00	8000.00
11	96.00	1096.00	7000.00
12	84.00	1084.00	6000.00
13	72.0	1072.00	5000.00
14	60.00	1060.00	4000.00
15	48.00	1048.00	3000.00
16	36.00	1036.00	2000.00
17	24.00	1024.00	1000.00
18	12.00	1012.00	.00

（下换一页）

NAME：TAN

SCHEDULE OF PAYMENTS

ORIGINAL AMOUNT $20,000.00

MONTH	INTEREST	TOTAL PAYMENT	UNPAID BALANCE
1	240.00	1906.67	18333.33
2	220.00	1886.67	16666.66
3	200.00	1866.67	15000.00
4	180.00	1846.67	13333.33
5	160.00	1826.67	11666.66
6	140.00	1806.67	10000.00
7	120.00	1786.67	8333.33
8	100.00	1766.67	6666.66
9	80.00	1746.67	5000.00
10	60.00	1726.67	3333.33
11	40.00	1706.67	1666.66
12	20.00	1686.67	.00

（下换一页）

NAME:CHANG

SCHEDULE OF PAYMENTS

ORIGINAL AMOUNT $12,000.00

MONTH	INTEREST	TOTAL PAYMENT	UNPAID BALANCE
1	144.00	644.00	11500.00
2	138.00	638.00	11000.00
3	132.00	632.00	10500.00
4	126.00	626.00	10000.00
5	120.00	620.00	9500.00
6	114.00	614.00	9000.00
7	108.00	608.00	8500.00
8	102.00	602.00	8000.00
9	96.00	596.00	7500.00
10	90.00	590.00	7000.00
11	84.00	584.00	6500.00

12	78.00	578.00	6000.00
13	72.00	572.00	5500.00
14	66.00	566.00	5000.00
15	60.00	560.00	4500.00
16	54.00	554.00	4000.00
17	48.00	548.00	3500.00
18	42.00	542.00	3000.00
19	36.00	536.00	2500.00
20	30.00	530.00	2000.00
21	24.00	524.00	1500.00
22	18.00	518.00	1000.00
23	12.00	512.00	500.00
24	6.00	506.00	.00

(下接一页)

NAME: LI

SCHEDULE OF PAYMENTS

ORIGINAL AMOUNT $45,120.00

MONTH	INTEREST	TOTAL PAYMENT	UNPAID BALANCE
1	541.44	1794.77	43866.66
2	526.40	1779.73	42613.33
3	511.36	1764.69	41360.00
4	496.32	1749.65	40106.66
5	481.28	1734.61	38853.33
6	466.24	1719.57	37600.00
7	451.20	1704.53	36346.66
8	436.16	1689.49	35093.33
9	421.12	1674.45	33840.00
10	406.08	1659.41	32586.66
11	391.04	1644.37	31333.33
12	376.00	1629.33	30080.00
13	360.96	1614.29	28826.66
14	345.92	1599.25	27573.33
15	330.88	1584.21	26320.00
16	315.84	1569.17	25066.66
17	300.80	1554.13	23813.33
18	285.76	1539.09	22560.00
19	270.72	1524.05	21306.66
20	255.68	1509.01	20053.33
21	240.64	1493.97	18800.00
22	225.60	1478.93	17546.66
23	210.56	1463.89	16293.33
24	195.52	1448.85	15040.00
25	180.48	1433.81	13786.66
26	165.44	1418.77	12533.33
27	150.40	1403.73	11280.00
28	135.36	1388.69	10026.66

29	120.32	1373.65	8773.33
30	105.28	1358.61	7520.00
31	90.24	1343.57	6266.66
32	75.20	1328.53	5013.33
33	60.16	1313.49	3760.00
34	45.12	1298.45	2506.66
35	30.08	1283.41	1253.33
36	15.04	1268.37	.00

(下换一页)

NAME: LING

SCHEDULE OF PAYMENTS

ORIGINAL AMOUNT $1,500.00

MONTH	INTEREST	TOTAL PAYMENT	UNPAID BALANCE
1	18.00	168.00	1350.00
2	16.20	166.20	1200.00
3	14.40	164.40	1050.00
4	12.60	162.60	900.00
5	10.80	160.80	750.00
6	9.00	159.00	600.00
7	7.20	157.20	450.00
8	5.40	155.40	300.00
9	3.60	153.60	150.00
10	1.80	151.80	.00

(下换一页)

NAME: FUNG

SCHEDULE OF PAYMENTS

ORIGINAL AMOUNT $5,000.00

MONTH	INTEREST	TOTAL PAYMENT	UNPAID BALANCE
1	60.00	893.33	4166.66
2	50.00	883.33	3333.33
3	40.00	873.33	2500.00
4	30.00	863.33	1666.66
5	20.00	853.33	833.33
6	10.00	843.33	.00

注：由于印刷条件的限制，以上打印位置只是近似地表示，而不是完全准确的。

【例 6.5】 有4个营业组，每日销售5种商品，要求将一周中每日的每一种商品销售数据进行处理。

每一种商品销售数据包括商品号、单价、本日售出数量。要统计出每个营业组每天的销售总额。

星期 / 销售值 / 商品号 / 营业组	星期一					星期二					星期三					星期四					星期五					星期六					星期天				
	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
一																																			
二																																			
三																																			
四																																			

共应读入4×7×5=140条商品记录。读完第一营业组的星期一的五个商品记录并处理后才读第一营业组星期二的五个商品。……

```
DENTIFICATION DIVISION.（标识部）
PROGRAM-ID. EXAM65.
ENVIRONMENT DIVISION.（环境部）
CONFIGURATION SECTION.
SOURCE-COMPUTER. VAX-11/750.
OBJECT-COMPUTER. VAX-11/750.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT  INPUT-FILE  ASSIGN  TO  IN-FILE.（输入文件）
    SELECT  LP-FILE ASSIGN TO P-FILE.（打印文件）
DATA DIVISION.（数据部）
FILE SECTION.（文件节）
FD  INPUT FILE
    LABEL  RECORD  IS  STANDARD
    DATA  RECORD  IS  GOODS-RECORD.
01  GOODS-RECORD0.（商品记录）
    02  GOODS CODE PIC  9(3).（商品代码）
    02  PRICE        PIC  999V99.（单价）
    02  QUANTITY   PIC  999.（数量）
FD LP-FILE
     LABEL  RECORD  IS  STANDARD
     DATA  RECORD  IS  PRINT-LINE.
01  PRINT-LINE.  （输出记录）
    02  FILLER        PIC  X(5).
    02  GROUP-P       PIC  9.（小组）
    02  FILLER        PIC  X(5).
    02  DAY-P         PIC  9.（日）
    02  FILLER        PIC  X(5).
    02  AMOUNT-P      PIC  $(6).99(金额)
WORKING STORAGE      SECTION.（工作单元节）
77  GROUP-W          PIC  9.（小组）
77  DAY-W            PIC  9.（日）
77  GOODS-W          PIC  9.（商品）
77  SALES-VALUE      PIC  9(5)V99.（销售额）
77  AMOUNT           PIC  9(5)V99  VALUE  ZERO.（金额）
PROCEDURE             DIVISION.（过程部）
BEGIN.
    OPEN  INPUT  INPUT-FILE
          OUTPUT  LP-FILE.
PERFORMING.
    PERFORM  DATA-PROCESSING
          VARYING  GROUP-W  FROM  1  BY  1  UNTIL  GROUP-W > 4
          AFTER    DAY-W    FROM  1  BY  1  UNTIL  DAY-W > 7
          AFTER    GOODS-W  FROM  1  BY  1  UNTIL  GOODS-W > 5.
       PERFORM  WINDING-UP.
DATA-PROCESSING.
       READ  INPUT-FILE
             AT  END  GO  TO  WINDING-UP.
       COMPUTE  SALES-VALUE = QUANTITY * PRICE.
       ADD  SALES-VALUE  TO  AMOUNT.
       IF  GOODS-W  NOT<5
```

```
            MOVE  SPACE  TO  PRINT-LINE
            MOVE  GROUP-W  TO  GROUP-P
            MOVE  DAY-W  TO  DAY-P
            MOVE  AMOUNT  TO  AMOUNT-P
            WRITE  PRINT-LINE  AFTER  1.
            MOVE  0  TO  AMOUNT.
WINDING-UP.
          CLOSE  INPUT-FILE, LP-FILE
          STOP  RUN.
```

如果在运行时从 IN-FILE.DAT 中读入数据如下：

```
商品号| 单价 |数量
 001  | 01012 |010 ┐
 002  | 00856 |008 │
 003  | 02432 |012 ├ 第一组五个商品的销售数据
 004  | 03820 |003 │ (星期一)
 005  | 00110 |007 ┘
 006  | 12032 |020 ┐
 007  | 11023 |005 │
 008  | 02200 |004 ├ 第一组五个商品的销售数据
 009  | 01200 |080 │ (星期二)
 010  | 02013 |100 ┘
       ⋮          ┐
       ⋮          ├ 第一组五个商品的销售数据
                  │ (星期三)
```

流程图见图6.15，其中图(b)是 DATA-PROCESSING 段的流程图。

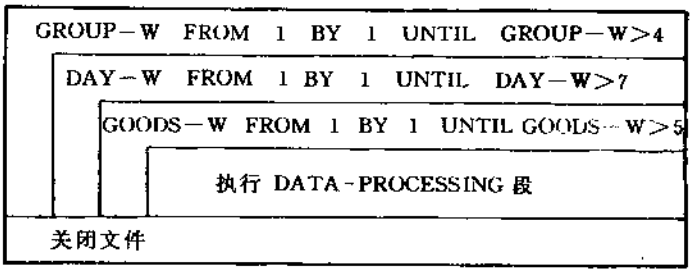
本程序使用了具有三重循环的执行语句。数据项 GROUP-W(营业组)为最外层循环变量，DAT-W(星期)为第二层循环的循环变量，GOODS-W(商品)为内层循环的控制变量。每次读入一记录后，即计算出该商品的销售额(SALES-VALUE)，并将它累加到当天该小组的总销售额(AMOUNT)中。直到读入五个记录，并计算出总销售额(该小组星期一的总销售额)。然后输出小组号，星期几，总销售额。并使 AMOUNT 置零，以使统计下一天的总销售额时初值为0。如此，每读五个记录输出一次结果，一共输出的行数为4×7＝28行。输出的格式为：

```
(营业小组号)   (星期几)   (总销售额)
     1            1      $5838.20 ┐
     1            2      $6018.55 ├ 第一组一周中每天销售额
     ⋮            ⋮          ⋮    ⋮
     ⋮            ⋮          ⋮    ⋮
     1            7      $653.24  ┘
     ⋮            ⋮          ⋮    ⋮
     4            6      $8121.28 ┐
     4            7      $534.86  ┘ 第四组一周中每天销售额
```

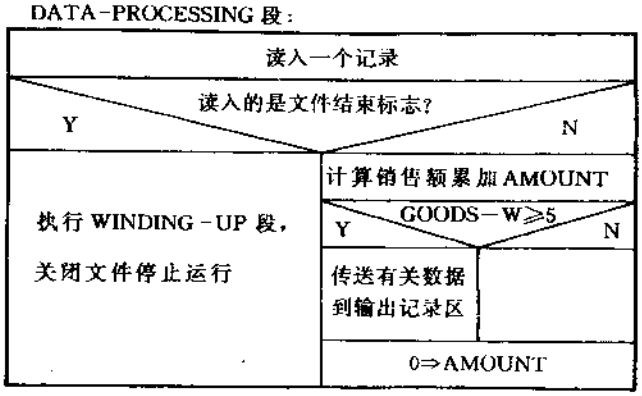
读者也可以修改此程序，输出表头说明。

说明：

(1) 在本例中在 READ 语句中的 AT END 子句中用 GO TO 语句，它转到本段之外，



(a)



(b)

图 6.15

而本段又是一个为 PERFORM 语句所调用的该语句序列，这是否可以？我们前面提到过一般不应采用这种用法，以免程序运行时出错。应该通过此语句序列的最后一语句返回到 PERFORM 的下一句。此例中，它转出去后就结束运行，而不要求返回，因此这样用是可以的。

(2) 每小组每天销售的商品的号码可以是相同的(例如1～5)。也可以是任意五种商品(如第一天1～5，第二天6～10，…或任何商品的号)只要在记录中给出各商品号和其单价、数量即可。由于本例是练习程序，故把问题简化了，也可把程序修改为100种商品，200种商品等等。

习　题

6.1　有哪几种形式的 PERFORM 语句？请各写出一个具体的 PERFORM 语句。

6.2　如有下列 A1到 A7段，问下面(1)～(8)的用法合法吗？如果合法，请顺序写出被执行的每一段的段名。

```
A1.  …
A2.  PERFORM  A4  THRU  A5.
A3.  …
A4.  PERFORM  A6.
```

A5. …
A6. …
A7. …

(1) PERFORM A1 THRU A5.
(2) PERFORM A4 THRN A6.
(3) PERFORM A5 THRU A7.
(4) PERFORM A2 THRU A6.
(5) PERFORM A4 THRU A5.
(6) PERFORM A2.
(7) PERFORM A2 THRU A1.
(8) PERFORM A2 THRU A3.

6.3 下面的 PERFORM 语句应执行几次 T 段?

```
(1) PERFORM T VARYING X FROM 1 BY 1 UNTIL X > 5.
(2) PERFORM T VARYING I FROM 2 BY 2 UNTIL I > 9.
(3) PERFORM T VARYING J FROM 10 BY -1 UNTIL J < 0.
(4) PERFORM T VARYING K FROM 8 BY -3 UNTIL K < -10.
(5) PERFORM T VARYING I FROM 1 BY 1 UNTIL I > 5
                AFTER J FROM 1 BY 1 UNTIL J > 5.
(6) PERFORM T VARYING I FROM 2 BY 2 UNTIL I > 10
                AFTER J FROM 2 BY 2 UNTIL J > 10.
(7) PERFORM T VARYING I FROM 1 BY 1 UNTIL I > 10
                AFTER J FROM 1 BY 1 UNTIL J > 10
                AFTER K FROM 2 BY 2 UNTIL K > 10
(8) PERFORM T VARYING X FROM 3 BY 3 UNTIL X > 10
                AFTER Y FROM 4 BY 3 UNTIL Y > 15
                AFTER Z FROM 5 BY 2 UNTIL Z > 20.
```

6.4 修改§6.10中例6.3使程序:

(1) 增加打印表头标题。

(2) 将过程部中的对应传送改为一个 MOVE 语句传送一个初等项。

6.5 学生成绩数据形式如下:班级、姓名、语文、数学、物理、化学、英语。假设学校某年级有四个班,编写程序分班打印学生成绩单。打印格式要求:

(1) 每班成绩单前有表头和标题。

(2) 除打印以上全部原始数据外,求出学生平均分,并把它打印出来。